2002-06

A computational model and multi-agent simulation for information assurance

VanPutte, Michael A.

Monterey, California. Naval Postgraduate School.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

A COMPUTATIONAL MODEL AND MULTI-AGENT SIMULATION FOR INFORMATION ASSURANCE

by

Michael VanPutte

June 2002

Dissertation Supervisor: Cynthia Irvine

This dissertation was completed in cooperation with the Institute for Information Superiority and Innovation and the MOVES Institute.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY	**2. REPORT DATE** June 2002	**3. REPORT TYPE AND DATES COVERED** Doctoral Dissertation	
4. TITLE AND SUBTITLE: A Computational Model and Multi-Agent Simulation for Information Assurance			**5. FUNDING NUMBERS**
6. AUTHOR Michael A. VanPutte			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Chief of Naval Operations, N6 2000 Navy Pentagon Washington, D.C. 20350-2000			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

ABSTRACT

The field of information assurance (IA) is too complex for current modeling tools. While security analysts may understand individual mechanisms at a particular moment, the interactions among the mechanisms, combined with evolving nature of the components, make understanding the entire system nearly impossible.

This dissertation introduces a computational model of IA called the Social-Technical Information Assurance Model (STIAM). STIAM models organizations, information infrastructures, and human actors as a complex adaptive system. STIAM provides a structured approach to express organizational IA issues and a graphical notation for depicting the elements and interactions. The model can be implemented in a computational system to discover possible adaptive behavior in an IA environment. A multi-agent simulation is presented that introduces several innovations in multi-agent systems including *iconnectors*, a biologically inspired visual language and mechanism for inter-agent communications.

The computational model and simulation demonstrate how complex societies of autonomous entities interact. STIAM can be implemented as a hypothesis generator for scenario development in computer network defensive mechanisms.

14. SUBJECT TERMS information assurance, information security, computer security, security model, modeling, agents, multi-agent system, multi-agent simulation			**15. NUMBER OF PAGES** 198
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release, distribution is unlimited.

A COMPUTATIONAL MODEL AND MULTI-AGENT SIMULATION FOR INFORMATION ASSURANCE

Michael A. VanPutte
Major, United States Army
B.S., The Ohio State University, 1988
M.S., University of Missouri – Columbia, 1997

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
from the

NAVAL POSTGRADUATE SCHOOL
June 2002

Author: _____
Michael A. VanPutte

Approved by:

_____	_____
Cynthia Irvine	Michael Zyda
Professor of Computer Science	Professor of Computer Science
Dissertation Supervisor	Dissertation Committee Chair
_____	_____
John Hiles	Rudy Darken
Professor of Computer Science	Professor of Computer Science
_____	_____
Neil Rowe	Don Brutzman
Professor of Computer Science	Associate Professor of Applied Science

Approved by: _____
Chris Eagle, Chair, Department of Computer Science

Approved by: _____
Carson Eoyang, Associate Provost for Instruction

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The field of information assurance (IA) is too complex for current modeling tools. While security analysts may understand individual mechanisms at a particular moment, the interactions among the mechanisms, combined with evolving nature of the components, make understanding the entire system nearly impossible.

This dissertation introduces a computational model of IA called the Social-Technical Information Assurance Model (STIAM). STIAM models organizations, information infrastructures, and human actors as a complex adaptive system. STIAM provides a structured approach to express organizational IA issues and a graphical notation for depicting the elements and interactions. The model can be implemented in a computational system to discover possible adaptive behavior in an IA environment. A multi-agent simulation is presented that introduces several innovations in multi-agent systems including *iconnectors*, a biologically inspired visual language and mechanism for inter-agent communications.

The computational model and simulation demonstrate how complex societies of autonomous entities interact. STIAM can be implemented as a hypothesis generator for scenario development in computer network defensive mechanisms.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. HYPOTHESIS 1
B. INTRODUCTION 1
C. MOTIVATION 2
1. Complex Adaptive Systems, Agents, and Multi-Agent Simulations 2
2. Multi-Agent Simulation of Information Assurance 3
D. APPROACH 4
E. CONTRIBUTIONS OF THIS WORK 4
F. DISSERTATION ORGANIZATION 5

II. REVIEW OF RELATED WORK 7
A. INTRODUCTION 7
B. INFORMATION ASSURANCE 7
C. MODELS AND SIMULATIONS OF INFORMATION ASSURANCE 8
1. Theoretical Models 8
2. Empirical Models 9
3. Computational Models 10
4. Miscellaneous Models 12
5. IA Attacker Taxonomies and Motivations 13
6. Security Taxonomies 16
7. Failures of Traditional Models 16
D. COMPUTATIONAL ANALYSIS OF INFORMATION ASSURANCE 18
1. Symbolic Approach – Rule-Based Systems 18
2. Connectionist Approach – Artificial Neural Networks 20
3. System Dynamics – Stochastic Simulations 21
4. Multi-Agent Simulations (MAS) 22
E. UNIFIED MODELING LANGUAGE 25
F. SUMMARY 26

III. COMPUTATIONAL MODEL OF INFORMATION ASSURANCE (IA) 27
A. INTRODUCTION 27
B. OVERVIEW 27
C. SOCIETY 28
D. DOMAINS AND ELEMENTS IN A SOCIETY 30
1. Tokens 30
2. Infrastructures 31
3. Organizations 36
4. Organizational Roles 39
5. Actors 42
E. SUMMARY 44

IV. CONNECTORS AND A CONNECTOR-BASED MODEL OF INFORMATION ASSURANCE 45
A. INTRODUCTION 45

B. ICONNECTORS 45
1. Introduction 45
2. Definitions 46
3. Iconnectors and Entity Interfaces 47
4. Formalism 47
C. ICONNECTOR GRAPHICAL NOTATION 49
D. ICONNECTOR COMPONENTS 50
1. Iconnector State – Extended or Retracted 50
2. Iconnector Types – Sockets and Plugs 51
3. Iconnector Cardinality 52
4. Iconnector Labels 53
5. Listening Iconnector 53
5. Actions 54
E. SOCKET AND PLUG CONNECTIONS 57
F. SUMMARY 61

V. THE STIAM CONNECTOR-BASED AGENT ARCHITECTURE 63
A. INTRODUCTION 63
B. OVERVIEW 63
C. CONNECTOR-BASED AGENT ARCHITECTURE 64
D. CONNECTORS AND THE INNER ENVIRONMENT 66
E. ROLE SET – R_I 68
F. GOALS - G_I 68
1. Goal Structure 69
2. Goal Manager 73
3. Action Set – Tickets and Frames 74
G. AGENT TOKENS AND KNOWLEDGE SET-- CAPABILITIES 77
1. T_i – Token Set 77
2. K_i – Knowledge Set 78
3. Agent Learning 78
H. BEHAVIOR MODERATORS 78
1. Observable Personality 79
2. Skills 79
3. Emotional State 79
I. LIMITATIONS OF STIAM AGENTS 80
J. SUMMARY 80

VI. MODEL VALIDATION 81
A. INTRODUCTION 81
B. INFORMATION ASSURANCE AND HYPOTHESES GENERATORS 81
1. Models and Simulations 81
2. Induction and Hypothesis Generation 82
3. STIAM 83
C. EMPIRICAL MODEL OF INFORMATION ASSURANCE (IA) 83
1. CERT/CC 83
2. Enhanced Model 86

D. MAPPING OF EMPIRICAL MODEL TO STIAM 88
1. Actors and Objectives 88
2. Tools 88
3. Vulnerability 90
4. Action 92
5. Target 98
6. Result 98
7. Summary 99
E. INFORMATION ASSURANCE (IA) AS A CONCURRENT SYSTEM 100
F. SUMMARY 102
VII. EXAMPLE SOFTWARE IMPLEMENTATION 103
A. INTRODUCTION 103
B. SOFTWARE IMPLEMENTATION 103
1. SimSecurity Package 105
2. Entity Package 107
3. Actor Package 108
4. Connector Package 113
5. Scenarios Package 115
6. Utilities Package 116
C. SUMMARY 117
VIII. SCENARIO IMPLEMENTATION 119
A. INTRODUCTION 119
B. SCENARIO ONE – "ADAPTIVE ATTACKER" 119
1. Background 119
2. Implementation 120
3. Experimental results of Scenario One 125
C. SCENARIO TWO – WINDOW OF VULNERABILITY 129
1. Background 129
2. Implementation 130
3. Experimental results of Scenario Two 138
D. OBSERVATIONS 148
1. Model Granularity 148
2. Visualization of Large Societies 149
E. SUMMARY 150
IX. CONCLUSIONS AND RECOMMENDATIONS 151
A. CONCLUSIONS 151
B. RECOMMENDATIONS FOR FUTURE WORK 152
1. Agent History 152
2. Behavior Moderators 152
3. Dynamic Role Assignment Assignments and Organizations. 152
4. Generating Tickets, Frames, and Actions at Runtime 153
5. Agent Learning 153
6. Complex Agent Goal Assignments 153

7. Discretionary Access Control Policies in the STIAM Model 153
C. SUMMARY 154
LIST OF REFERENCES 155
GLOSSARY 165
APPENDIX A – EXECUTION OUTPUT 169
APPENDIX B – UML QUICK REFERENCE 173
INITIAL DISTRIBUTION LIST 175

LIST OF FIGURES

Figure 1. Spectrum of agent architectures. 24
Figure 2. A Conceptual Diagram: A Society composed of 29
Figure 3. A Token class in UML 30
Figure 4. An Infrastructure composed of Resources, Interfaces, and Tokens. 32
Figure 5. UML diagram of an Interface. 33
Figure 6. An Organization consisting of Roles and Policies 37
Figure 7. A Role and its components. 40
Figure 8. Multiple homogeneous System Administrator roles. 42
Figure 9. Conceptual diagram of an *actor* entity. 44
Figure 10. The IBinder *binds* Iconnectors, which are components of Entities. 47
Figure 11. Iconnector Class Diagram. 48
Figure 12. An Actor 'bob', an Infrastructure 'proprietary_network with a Resource 'corp_database' and an Organization 'enterprise' 49
Figure 13. Entities with Iconnectors added. 50
Figure 14. Extended and Retracted Iconnectors for an Infrastructure. 51
Figure 15. Socket and Plugs depicted graphically. 52
Figure 16. Socket cardinality diagramming convention. 53
Figure 17. An Actor Plug attempting to bind to an Infrastructure Socket Iconnector. 53
Figure 18. Listening Iconnectors. 54
Figure 19. Actor a_k binding to an Iconnector and a different 56
Figure 20. Connectors that permit Token ti or tj. 56
Figure 21. An Infrastructure and an Actor with *transfer* messages. 57
Figure 22. A sequence diagram of Socket and Plug connecting and 60
Figure 23. Agents and Objects operating in an external environment. 63
Figure 24. The components of a connector-based agent and their interactions. 66
Figure 25. Connector-based agent in an external environment. 67
Figure 26. Internal components with connectors extended into e_i. 68
Figure 27. A goal receives input from e_i and the outer environment, and produces a state, measure, and actions that effect the outer environment. 70
Figure 28. STIAM goal trigger and reset thresholds. 71
Figure 29. Actor goal state transitions. 72
Figure 30. A snap-shot of a typical actor's goals. 74
Figure 31. An example ticket. 75
Figure 32. A Ticket tk_i can dynamically bind to Actions j, but is not able to bind to action i. 77
Figure 33. Howard's *Computer and Network Attack Taxonomy*, [Howard, 1997]. 84
Figure 34. Howard and Longstaff's Computer Security Incidents [Howard and Longstaff, 1998]. 87
Figure 35. A socket that represents *flooding* a *resource*. Successfully binding to a flood disconnects (retracts) other iconnectors. 93
Figure 36. An Iconnector that replicates the *authentication* process. 94
Figure 37. Bypass Actions in STIAM. 95
Figure 38. Deletion and backup on STIAM. 97
Figure 39. The package diagram for an implementation of STIAM. 105

Figure 40. SimManager builds the GUI, loads a scenario, and repetitively loops through all of the Actors. 106
Figure 41. Scenario loading activities for SimManager class. 107
Figure 42. The `entity` package contains the `Entity` class and two specialized passive entities: `Infrastructure` and `Resource`. 108
Figure 43. The main classes in the actor package are the `Actor` and `CompositeAgent` classes, which inherit from the `Entity` class. 109
Figure 44. An activity diagram representing an agent goal selection routing. 110
Figure 45. An activity diagram depicting the agent reevaluate goal routine. 111
Figure 46. The classes of the `actor` package. 112
Figure 47. The classes related to `IConnectors` in the `connector` package. 114
Figure 48. The classes related to `Connectors` in the `connector` package. 115
Figure 49. Sample XML scenario file. 116
Figure 50. Classes in the `utilities` package. 117
Figure 51. An `enterprise` *infrastructure*, with a *resource*, service scan, and vulnerability 121
Figure 52. The `library` infrastructure, which provides information on the `enterprise` infrastructure. 122
Figure 53. The hackerSite infrastructure, which provides vuln103 Token if presented with sysType Token. 122
Figure 54. The example hacker's goals, tickets and actions. 124
Figure 55. Screen shot of Scenario One on STIAM implementation. 125
Figure 56. The elite (a) and script (b) infrastructures are identical except for the socket labels. 132
Figure 57. The enterprise infrastructure has an alert plug, a vulnerability socket, and a patch socket. 132
Figure 58. The vendor infrastructure represents the entire vendor community. 133
Figure 59. The Attacker role consists of three goals: acquire, exploit, and publish vulnerabilities. 134
Figure 60. The system administrator role. 135
Figure 61. Implementation of Scenario Two. 137
Figure 62. Typical exploit distribution graph shape reported by Arbaugh et al., 2000. 138
Figure 63. Results of reactive system administrators with patch released after scripts. 140
Figure 64. Reactive system administrator with accelerated publication of script and delayed publication of patch. 141
Figure 65. Reactive system administrator with patch released prior to scripts. 142
Figure 66. Results with proactive system administrators with patch released after scripts. 143
Figure 67. Proactive system administrators with scripts released soon after the elites and the before the patch. 144
Figure 68. Patch released prior to publication of exploit on script kiddie infrastructure for proactive system administrators. 145
Figure 69. Society with large number of attackers than infrastructures; using reactive system administrators. 146
Figure 70. Example deaggregated organization. 149

LIST OF TABLES

Table 1. Cohen's Threat Actors [From Cohen 2000]. 15
Table 2. Mapping of Howard and Longstaff's tools to STIAM. 89
Table 3. Comparison of key components in Howard and Longstaff model and the STIAM model. 100
Table 4. The Tokens used in Scenario One. 120
Table 5. Sequence of steps used by `Hacker` to access the critical resource. 126
Table 6. The Tokens used in Scenario Two. 131
Table 7. Results of Window of Vulnerability Scenario 147

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF EQUATIONS

Equation 1. A Society of Organizations, Infrastructures and Actors. 29
Equation 2. A Token is an element in the set of all possible Tokens..... 30
Equation 3. An Infrastructure composed of Information Resources,..... 31
Equation 4. The components of the interface..... 33
Equation 5. An Organization consisting of Roles and Policies..... 36
Equation 6. A Policy consisting of an Entity, Infrastructure, Mode, and Authorization. 38
Equation 7. A Role consists of Role Requirements, Role Goals, and Tokens..... 40
Equation 8. Role Requirements as a Collection of Sets..... 41
Equation 9. An iconnector specification consisting of Labels, State, and Cardinality. ... 48
Equation 10. The binding of a Socket to a Plug iconnector..... 52
Equation 11. A Resource Action definition. 55
Equation 12. A Connection Action definition..... 55
Equation 13. The Connector-Based Agent Specification..... 65
Equation 14. Connectors consist of a Label and State. 67
Equation 15. A goal definition. 69
Equation 16. Howard and Longstaff's Functional Model..... 101

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Many people contributed to this dissertation. I would like to thank my committee for their guidance and insight. I would especially like to thank Dr. Michael Zyda for his inspiration and direction along this journey. In addition, I would like to thank Dr. Cynthia Irvine for her continued efforts in passing on her knowledge and vision in the security field. Also, to Don Brutzman, for passing on his wisdom on what it is to be a computer scientist.

This dissertation would not have been possible without the intellect of Professor John Hiles. His insight and innovation in biologically inspired multi-agent systems provided the foundation *and* mechanisms upon which this dissertation is built.

This work would not have been completed without the support of the faculty, staff, and students of the Naval Postgraduate School. Special thanks goes to Dan Warren, Tim Levin, Michael Thompson, John Falby, Michael Capps, and Nelson Irvine.

I am indebted to my compatriots in this trek, the fellow doctoral students who have all contributed to the research: Brian Osborn, Curt Blais, Joerg Wellbrink, Simon Georger, and Perry McDowell.

Finally, I would like to thank my wife Linda, and daughters Ashley and Brianne, whose love and support kept me going along this journey. It is to them that this dissertation is dedicated.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

"The beginning of knowledge is the discovery of something we do not understand."

-- Frank Herbert

A. HYPOTHESIS

The information assurance domain at the organizational level is a dynamic, highly connected social and technical system. Modeling this domain as a multi-agent system can capture all of the key elements and interactions in the domain. Implementing this model as a software system can generate validatable hypotheses of the IA domain.

B. INTRODUCTION

Information Assurance (IA) is concerned with protecting and defending information and information systems [NSTISSC, 2000][1]. The field is complex and deals with highly interconnected social and technical components. In an attempt to understand and explain portions of the domain, researchers have developed various security models and simulations. While sufficient for the purpose for which they were designed, these tools provide limited utility for researchers to infer general conclusions at the organizational level.

The environment of IA is too complex and dynamic to be understood with our present tools. While IA researchers and security analysts may understand the individual mechanisms at a particular moment, the interactions that take place among the mechanisms, combined with the constantly evolving components themselves make understanding the entire system nearly impossible. Researchers do not have a *computational model* suitable for simulation of the domain that includes the numerous actors, objects, processes and interactions in the environment. Instead, analysts are

[1] Many terms used in the fields of information assurance *and* agent-based systems are ambiguous or defined in multiple ways. A glossary is provided at the end of this dissertation that defines the key terms used in this work.

forced to focus on pieces of the problem without being able to envision the global environment within which they are working.

This dissertation introduces a *computational model* of the IA domain. The model proposes, at a high level of functional abstraction, the actors, objects, and processes that interact in the IA domain. An extensible multi-agent simulation (MAS) is provided as an implementation of that model. To implement this model as a computational simulation, various innovations in multi-agent simulations are introduced. This computational model and simulation demonstrate how complex societies of highly interactive, autonomous actors and systems can combine and the security implications resulting from their interactions and combinations.

This dissertation also presents a graphical and mathematical notation for expressing the IA issues of an organization. While this notation can present the instantaneous issues at a point in time, its true benefit is to view the IA issues of an organization as they evolve. These graphical and mathematical notations permit IA analysts to view the dynamic nature of IA in an organization.

C. MOTIVATION

1. Complex Adaptive Systems, Agents, and Multi-Agent Simulations

The purpose of simulations is to facilitate scientific study of *complex systems*. A *model* is an abstraction of real world objects and processes that captures key aspects in the *system* under investigation. A *simulation* is an implementation of the model. Models and simulations permit researchers to investigate real-world systems and perform experiments that are not possible in the real systems [Law and Kelton, 2000].

If the relationships among the objects and processes in the system are relatively simple, then mathematics may provide an analytical solution. For domains that are more complex, a simulation can be used to provide insight into the model and real-world system [Law and Kelton, 2000].

Some real-world environments are complex adaptive systems (CAS), systems involving nonlinear relationships among large numbers of highly connected, interacting, adaptable entities. Due to the complexity of these systems, mathematical tools and

traditional simulations often cannot accurately represent these domains. Artificial complex adaptive systems (ACAS), composed of autonomous, interactive software *agents* are more capable for simulating such complex systems [Axelrod, 1997], [Holland, 1996]. This work applies these capabilities to the field of IA.

2. Multi-Agent Simulation of Information Assurance

IA is concerned with "…protect(ing) and defend(ing) information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation" [NSTISSC, 2000]. The overall *system* is not restricted to technological components. A system is "a collection of entities, e.g., people or (and) machines, that act and interact together towards accomplishment of some logical end." [Law and Kelton, 2000]. Clearly, IA includes human actors that interact within this system, and any simulation that claims to model IA at the organizational level must include human aspects of the problem.

IA deals with adaptable humans and computational devices that are interconnected through webs of communications networks. Software and devices adapt through human interaction or autonomously to perform tasks. Humans adapt themselves, communication links, devices, and the software running on those devices, sometime unknowingly, to better achieve their goals. The domain is a interconnected, dynamic environment, where changes in one part of the environment can have cascading effects in other parts. For example: requiring long, complex passwords composed of alphanumeric and non-alphanumeric characters may cause legitimate system users to write passwords down and place them in unsecured locations, such as yellow sticky labels posted on a computer monitor or under a desk pad. The requirement for long passwords may reduce the threat of an attacker guessing a password, but may increase the threat of an insider *finding* and *using* the password, in effect mitigating the security enhancement that the long password was originally meant to achieve.

The developed architecture provides an environment where investigators can conduct research and gain insight into the area of IA. By developing a virtual IA laboratory, IA researchers can develop and view an abstraction of the domain, ask and

answer questions via MAS experiments and gain insight into actual strengths and vulnerabilities.

D. APPROACH

A generalized computational model of the IA domain was developed through study of previous models, simulations, and observations of trends in the IA field. A multi-agent simulation (MAS) has been developed that is an implementation of this model. The MAS was tested on various scenarios, and the output compared with real-world results. The results show that it is possible to simulate the IA domain as a CAS.

While an implementation may be configured to confirm or deny a hypothesis, the true power of the system is in its ability to discover patterns, providing insight into the possible evolutionary patterns of the environment, which can then be carefully confirmed or denied in the real world.

A validation of this model is provided by mapping the elements of an empirically based model of IA to this research. Additionally, a multi-agent simulation of this model was developed. The scenarios implemented were compared with results in the real world.

E. CONTRIBUTIONS OF THIS WORK

This dissertation provides a fundamental new approach to examine IA issues at the organizational level. This dissertation provides the following fundamental new contributions:

- A formal mathematical and graphical language for representing the entities and their interactions in organizational modeling of IA.
- An abstract computational model providing a mathematical depiction of the social and technical aspects of the actors, objects, and processes in the IA domain, and how these components interact.
- A descriptive model providing a graphical notation and semantics for depicting and visualizing IA environments.
- An extensible multi-agent architecture for the simulation of the IA environment.

- An extension of existing works on *connectors*, including both intra-agent and inter-agent communications, providing not only lightweight communication mechanisms, but also a graphical notation for visualizing communications among entities.
- An implementation of an innovative composite-agent architecture that takes advantage of the connector-based communications mechanism.

F. DISSERTATION ORGANIZATION

The remainder of this dissertation is organized as follows.

- Chapter II provides the reader with background on existing models and simulations in the IA domain and multi-agent simulation systems.
- Chapter III introduces a computational model of IA, presenting the model in formal mathematical notation and in the Unified Modeling Language [Booch *et al.*, 1999].
- Chapter IV introduces iconnectors, a graphical notation to illustrate communications among entities and a data structure to implement connector-based systems. The IA model presented in the previous chapter is presented using this connector notation.
- Chapter V presents a Connector-Based Agent Architecture [Hiles *et al.*, 2001] that was implemented for simulating humans throughout the simulation.
- Chapter VI provides an evaluation of the model, and discusses the advantage of this concurrent model over functional models of IA.
- Chapter VII describes a proof of concept software implementation of the multi-agent IA computational model.
- Chapter VIII discusses several scenarios that were implemented on the multi-agent software, and a corresponding analysis of the scenarios. This is followed by general observations discovered in the implementation of the model and scenarios.
- Chapter IX provides a discussion of future work, and conclusions.

- A glossary is provided for reader convenience.
- Appendix A provides a listing of output from implemented scenarios.
- Appendix B is a Unified Modeling Language (UML) Quick Reference, providing an overview of the UML notation used in this dissertation.

II. REVIEW OF RELATED WORK

"...you don't just solve problems, you defend against threats. If you study hard enough, you can understand a computer problem completely, because it's a matter of physics and electronics and software. The threat comes from a human attacker, not a machine...Security is not a technical problem, it's a social issue. If you treat it as a problem that can be solved by technological means, you leave yourself open for an attack."

-- Thomas Wadlow, The Process of Network Security

A. INTRODUCTION

This chapter provides an introduction to the information assurance (IA) domain, discusses challenges in modeling and simulating this domain, and presents previous models and simulations that have been developed. It then introduces alternative technologies available to simulation developers and discusses why multi-agent systems (MAS) are the best tool for modeling IA at the organizational level. Finally, it discusses the use of the Unified Modeling Language (UML) in this dissertation.

B. INFORMATION ASSURANCE

Information Assurance (IA) is concerned with "…protect(ing) and defend(ing) information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation" [NSTISSC, 2000]. This dissertation is primarily concerned with the issues of availability, confidentiality, and integrity of the information and information systems at the organizational level. These three characteristics of IA are defined in [NSTISSC, 2000] as:

- **availability**: "Timely, reliable access to data and information services,"
- **confidentiality**: "Assurance that information is not disclosed to unauthorized persons, processes or devices,"
- **integrity**: "…protection against unauthorized modification or destruction of data (and processes)."

These three characteristics are not independent, and may overlap and even conflict with one another. For example, strong confidentiality may adversely affect availability [Pfleeger, 1997].

C. MODELS AND SIMULATIONS OF INFORMATION ASSURANCE

1. Theoretical Models

Theoretical models are developed to help understand a complex system under investigation. Several researchers have attempted to create theoretical models of the IA field to help explain the environment. Modeling the entire domain is a vast undertaking, and "a comprehensive taxonomy in the field of computer security has been a relatively intractable problem" [Amoroso, 1994].

Numerous formal models have been developed to demonstrate various security principles. Bell and La Padula developed a Confidentiality Model to formally describe the Department of Defense Multilevel Security Policy, showing in abstract terms the authorized flows of information in secure systems [Bell and LaPadula, 1973]. This model uses formal mathematical notation to describe which actors and processes can read and write to an object in an abstract operating system.

The Biba Integrity Model [Biba, 1977] is based on the observation that the Bell and LaPadula model was only developed to deal with unauthorized disclosure of information. Biba examined the unauthorized modification of data, but ignored secrecy. Researchers are attempting to combine the security and integrity policies to form a more complete model.

Graham and Denning developed a formal model of protection that consisted of subjects, objects, rights, and an access control matrix [Graham and Denning, 1972]. This model provided the foundation for later models. The Harrison, Ruzzo and Ullman model [Harrison *et al.*, 1976], based on the Graham-Denning model, proved a fundamental limitation of automated examination of computer security systems. This model proved that "…it is not always *decidable* whether a given protection system can confer a given right" [Harrison *et al.*, 1976]. This conclusion implies that there is no algorithm that can prove that an arbitrary operating system will provide an arbitrary access to an arbitrary

data object. A system can be designed such that the access to information is decidable (every command of the operating system must be an atomic operation), but these systems may be restricted in functionality.

These formal state models are used to specify system protection behavior, such as access control or the prevention of information leakage. They model a policy, and a system implements that policy or it does not. While these models are helpful in understanding disclosure and modification of information in formal systems, they have limited utility in comprehensively modeling the entire domain of IA.

2. Empirical Models

Howard's dissertation was an analysis of Internet incidents from the Computer Emergency Response Team at Carnegie Mellon over the period 1989 to 1995 [Howard, 1997]. The model was based on data collected on Internet security incidents, and provides a useful first step in illustrating attacker intent, tools, and effects.

Howard and Longstaff updated Howard's model, providing additional coverage of security incidents based on their experience in the security field [Howard and Longstaff, 1998]. The model includes categories of attackers, tools, vulnerabilities, actions, targets, results, and attacker objectives. This model, based on empirical data and experience, is very useful for categorizing security incidents, and is discussed in detail in Chapter VI.

Building upon his foundational analysis of intentional and accidental misuse techniques [Neumann and Parker, 1989], Neumann [1995] provides a comprehensive discussion of the threats, vulnerabilities, and risks to computer systems based on data collected from 1976 to 1995. Neumann's analysis is more comprehensive than Howard's, including such categories as interpersonal attacks, accidents, and ignorance in his discussion of risks to computer systems. While he doesn't provide an overall model, he does provide a wealth of information upon which others may base their models.

Amoroso developed a cost-effects matrix [Amoroso, 1994] describing actors and possible actions, but not their reasoning. Landwehr's model [Landwehr *et al.,* 1994] is a partial classification of possible attack "mechanisms" that lacks details such as attacker's goals and possible countermeasures that defenders may employ. See [Cohen 2000] for a detailed discussion of these models.

3. Computational Models

The purpose of a computational model is to describe a domain with sufficient expressive detail that a computational simulation can be built based on the model. The simulation can provide insight into the field being investigated and verify the model.

a. Rowe and Schiavo

Rowe and Schiavo created a simulation to generate plans for software representations of legitimate users and cyber attackers as a component to an automated intrusion detection tutorial system [Rowe and Schiavo, 1998]. This planning tool used a modified means-ends analysis [Newell and Simon, 1972] to generate plans for entities to achieve goals. The simulation was a multi-agent system, with each entity in the simulation an autonomous software entity. The individual agents used a top-down planning approach to define the actor's plans and actions. Additionally, the system took into consideration time and probabilities, creating a realistic simulation of attacker and user behavior in simulated system logs. While the system produced realistic, intelligent behavior, it suffered from the same problems as all top-down rule-base systems; the engineer must predefine the rules, based on a belief that actors behave in a certain way. This reliance on predefined behaviors prohibits the agent from discovering innovative ways to deal with unforeseen situations. The strengths and weaknesses of rule-based systems are discussed in more detail in Section D.

b. Liu, Yu, and Mylopoulos

Liu *et al.* [2002] used the i* intentional framework [Yu, 1997] to analyze security requirements as a social system. The framework depicts an environment under investigation as a dependency graph among "actors" and their goals, providing a means to analyze multiple actors and the intentional dependencies between them. By examining the dependencies between supporting and conflicting actor goals, analysts are able to determine potential threats to systems.

The i* model focuses entirely on strategic relationships among actors. System vulnerabilities and exploits are generated "ad hoc" as a specific case and placed manually in the model. The i* model explores the relationships between actors at the

intentional level. As such, this framework allows analysts to determine "why" an actor chooses a course of action that may lead to a compromise.

First, detailed technical requirements are not modeled in the i* model, since Lie is interested in the relationships among actors and not the technical specifications of the system. Secondly, this dissertation is interested in modeling behavior, and not intention. It is interested in "how" a series of events may lead to a compromise. While this dissertation is not interested in cognitive modeling, the graphical nature of i* may provide a useful means to depict actor roles, goals, and actions in later work.

c. Cohen

Cohen provided the first detailed computational model and simulation for "simulating cyber attacks, defenses and consequences" [Cohen, 2000]. This simulation consisted of a database of 37 threat mechanisms, 94 attack mechanisms, and 140 protection mechanisms along with how these mechanisms are related and their effects. The database was used as input into a discrete event simulation. The simulator was run repeatedly and the output was statistically analyzed. In Cohen's model the "actors" were simple translation tables. Successfully accessing a node in a network resulted in the attacker being able to "pass through that node." Success for an attacker was defined as gaining access to a specific important node. It was not possible to show that a node was offline or *disabled*. Additionally, it was not possible to show interim benefits of compromising hosts. One such benefit is access to data files that can provide the attacker with additional useful information. Another benefit of penetrating a host is the additional computational capacity of the compromised host, for example, in a denial of service attack or a distributed brute force password-cracking scheme. While this simulation was an important first step, the system lacked actor adaptability as found in evolutionary social systems involving humans. Additionally the simulation does not permit multiple attackers or collaborating defenders, a necessity in simulations of a highly social yet adversarial environment like IA.

4. Miscellaneous Models

The following models and systems are various attempts to model components of the IA domain and develop computational systems based on these models. They are included to show various other techniques used to model components of the environment.

(a) Immunology Models

Forrest *et al.* [1996] developed a limited security model based on the natural immune system. She developed a limited intrusion-detection system that performed in a manner similar to the animal immune systems response to intrusions by disease. Her original solution to the problem led to promising results. Her work to model a technical system as a biological system was limited to monitoring privileged system calls, but the idea of modeling security as a biological system provides insight into developing other biological based systems.

(b) Information Warfare (IW) Models

Anderson [1998] examined risk assessment in the IW domain, attempting to model actual human threat actors in specific situations in order to apply resources to counter real threats. Anderson categorized threat actors based on whether they are enabled (have a capability to perform specific adversarial actions) and have access to systems, information, and personnel needed to perform these actions. Combining intent and motives provides a database of capable and motivated threats to systems. When analysts correlate actual indications of attackers, with actors who are motivated and enabled, they can make informed judgments regarding who is likely responsible for these indicators. Although Anderson was not developing this architecture for simulation, his model provides a starting point to model threat actors and intent in multi-agent systems.

(c) Network Analysis

The field of *network* modeling and simulation uses queueing models and protocol analysis to analyze changes to protocols, packet size and format, and network configurations to optimize system performance [Katzela, 1998]. Additional constraints have been added to some models, to investigate additional aspects of a network. Network Warfare Simulation (NETWARS) allows users to add additional constraints to a network simulation, to see the "...unanticipated effects of full operational combat network

loadings." The U.S. Department of Defense developed this communications modeling tool to "credibly model tactical communications demands with all the stresses and inefficiencies that combat places on communication systems" [DISA, 2001]. These models are only able to analyze the technical aspects of networks, and do not address social issues.

5. IA Attacker Taxonomies and Motivations

Numerous researchers have attempted to build taxonomies to classify attack actors.

Denning [1990] limited her analysis to "non-malicious hackers," or "someone that experiments with systems… playing with systems and making them do things that they were never intended to do" [Denning, 1990]. She developed five types of hacker motives:

- access to computers and information for learning,
- thrill, excitement, and challenge,
- ethics and avoiding damage,
- public image and treatment,
- privacy and first amendment rights.

This introduces the motives of one aspect of the human threat to information systems, but does not account for other aspects of human threats, such as *insiders* and malicious attackers.

Wadlow [2000] states, "Attackers will be successful if they have sufficient skill, motivation, and opportunity." He goes on to state that there are three categories of attackers:

- browsers, campers, and vandals;
- spies and saboteurs;
- and disgruntled (ex-) employees and (ex-) contractors.

Browsers, campers, and vandals are the stereotypical hackers/crackers. Browsers want to penetrate and look around. Campers penetrate to use the superior resources of the target, such as high-speed networks, processors, and memory. Vandals are often campers who have been discovered; they commit service denial or damage for ego

gratification. Browsers, campers, and vandals are typically *script kiddies* – inexperienced individuals who don't understand what they are doing, reusing script tools developed by others – motivated by ego and a desire to boast to friends. They are not interested in the target system itself, rather they are interested in the resources, and will try to exploit those if they believe they can get away with it. These attackers are not likely to use extraordinary means. Wadlow uses the metaphor of wasps to describe their behavior: they look for an easy way into a system, they are difficult and expensive to remove once in, attacking them is foolish and dangerous, and ignoring them means you can never use the resource [Wadlow, 2000].

Spies are looking for something very specific, where saboteurs want to deny you from doing something. There are very few spies and saboteurs but they have very high skills and are very determined. Spies may be political, freelance, or industrial. They typically target specific individuals, corporations, and government agencies. Their method is to collect huge amounts of information about the target system, rehearse before an actual attack, and if they are detected before they are finished, they will walk away. Saboteurs differ from vandals in that they target specific individuals, rather than vandals who target anyone. Saboteurs also have a higher goal driving their damage and denial operations.

Disgruntled (ex-)employees, and (ex-)contractors are motivated by being displeased. They have the skills, training, and experience on the equipment that will be targets. They have opportunity because they have access to systems, and their knowledge is high because they know the system capabilities and vulnerabilities that provide access.

Cohen [2000] provides the most comprehensive categorization, listing 37 categories (see Table 1 -- Cohen's Threat Actors). Each of these threats has a corresponding definition, and a very general discussion of their likely goals. These threats are also cross-linked to the attacks they are likely to use. Although the categorization was thorough, it is too general for sophisticated goal analysis and planning.

activists	foreign agents and spies	nature
club initiates	fraudsters	organized crime
competitors	global coalitions	paramilitary groups
consultants	government agencies	police
crackers for hire	hackers	private investigators
crackers	hoodlums	professional thieves
customers	industrial espionage experts	reporters
cyber-gangs	information warriors	terrorists
deranged people	infrastructure warriors	tiger teams
drug cartels	insiders	vandals
economic rivals	maintenance people	vendors
extortionists	military organizations	whistle blowers
	nation states	

Table 1. Cohen's Threat Actors [From Cohen 2000]).

Carroll [1995] analyzed computer crime using the acronym *MOMM* for Motives, Opportunity, Means, and Methods. Carroll discusses four motives for computer crimes; money, ideology, compromise (coercion), and egotism. Opportunity consists of technical knowledge and physical and electronic access of a potential attacker. Means are the processes used by the attacker to perform the attack; and are a general description of the action the attacker will do to achieve his goals, such as obtaining funds by printing a check or transferring funds to a location the attacker can access. The method is the technical tool used to achieve the means.

Parker [1998] used the acronym *SKRAM* (skills, knowledge, resources, authority, and motives) to differentiate cyber criminals based on properties or attributes that the criminals possess, not the activities they perform. Parker's categories of attacks include insider, malcontents, irrational, extremists, terrorists, personal problem-solver, cyber criminal, malicious hacker, hacker, and prankster. Although the analysis is useful, it is not sufficiently complete for a computational simulation. For example, the attacker's dedication is not considered to describe the amount of time an attacker may take before being frustrated and quitting, or the amount of risk an attacker might be willing to take to complete a mission.

These taxonomies provide insight on threat actor motivations, but they lack sufficient detail for building a comprehensive computational simulation of the IA

domain. Actors cannot be analyzed in a vacuum – and so at best, these categories provide a snapshot of a potential attacker at a moment in time. They may capture their skills and motivations at that moment, but they don't help explain the ultimate aims of the actor.

6. Security Taxonomies

There have been several attempts to build security taxonomies. In the development of the Common Criteria [NIST, 1999] a superficial security taxonomy was developed to help explain "security concepts and terminology... and relationships," and not as input into a simulation. It greatly simplified the attacker and their goals.

Several other security-related taxonomies have been introduced. Landwehr *et al.* [1994] proposed a comprehensive taxonomy of software security flaws. This research examined both "inadvertent" as well as "intentional" flaws, and built a high-level typing of software-introduced vulnerabilities. They also provided taxonomies based on 'time of introduction' and the location of the flaw introduction.

Victor Raskin is currently developing a security ontology, but it is currently incomplete, and no security simulations based upon this ontology have yet been developed [Raskin and Nirenburg, 2001].

7. Failures of Traditional Models

The major problem with the previous security models and simulations is that they fail to observe that information assurance is fundamentally both a technical and social problem and should be modeled appropriately. The U.S. Joint Chiefs of Staff state that *human actors are one of the basic sources of threats to information systems*. They go on to say that defeating information system threats requires the integration of *people*, operations, and technology [DoD, 2000]. People are a component of the system, yet people are not part of the models.

While few might argue that computer networks are social systems, little research has been conducted on the specifics of security-related social implications. Denning *et al*. point out that "If we ignore (the) social aspects (of computer security), there is the danger of developing technologies that are not cost effective, do not address the actual threat, or jeopardize human rights" [Denning *et al.*, 1987]. They go on to say that there

are “four topics related to the social aspects of computer security: security policy definition and awareness, user productivity, privacy, and information security.” While their paper was a call to the security community to consider these topics in the development of computer systems, it provides a good starting point in the consideration of social modeling to computer security.

Rheingold examined the social structure of “cyber villages.” He states, “One of the surprising properties of computing is that it is a social activity...” [Rheingold, 1993]. He goes on to say that the anonymity of the network permits you to extend your “circle of friends” who have shared values and interest, and that the circle of friends provides an "information social contract to share information, not based on reciprocity, but on a gift economy” [Rheingold, 1993].

When we look at empirical evidence of network attacks, we see social systems and deception in both attackers and defenders. In The Cuckoo's Egg, Cliff Stoll [1990] discovers KGB-sponsored hackers on his network. Stoll creates and nurtures a social group to defeat these attackers, and manipulates the attackers into performing acts (lengthy downloads from a ‘honey pot’) that result in their apprehension. While Stoll was somewhat more candid than his adversaries, his behavior was no less exploitive or manipulative than that of his adversary. In Masters of Deception, Slatalla [1995] provides insight into a social system of juvenile hackers. The social system permits the hackers to share information and learn how to exploit systems. This social system also results in their detection and downfall once their social group is penetrated.

These two examples illustrate that, although the field of IA involves sophisticated technology, it is very much related to traditional warfare, spycraft, and statecraft. Both attacker and defender are involved in various methods of intelligence, counter intelligence and deception operations. If one were able to apply Machiavelli’s observations on deceit and conspiracy [Machiavelli, 1515], malicious activities in the information security domain might no longer exist. Machiavelli observed that “the difficulties that confront a conspirator are infinite… many have been the conspiracies, but few have been successful; because he who conspires can not act alone, nor can he take a companion except from those whom he believes malcontent, and as soon as you have

opened your mind to a malcontent you have given him the material with which to content himself…" As Donath pointed out however, the attacker can hide his real identity. This anonymity and lack of fear of reciprocity by the "state" can motivate the attacker to perform network "conspiracies." Once an attacker's organization is penetrated then one can see the environment revert to Machiavelli's traditional model.

So a true simulation of the information assurance domain that covers both offensive and defensive capabilities, is a combined system that must model both human social interactions and the technical requirements which facilitate or constrain the social interactions.

D. COMPUTATIONAL ANALYSIS OF INFORMATION ASSURANCE

The ultimate purpose of modeling is to assess the "big picture" of a domain and gain insight into the inner workings of the system under investigation. Prietula states that "Organizations are complex, dynamic, non-linear, adaptive, and evolving systems" and analytical models are a poor choice for modeling these systems [Prietula *et al*., 1998]. Computational analysis is a useful tool for studying these systems, but *which* computational methodology is most appropriate for modeling information assurance at the organizational level?

In an attempt to answer this question, numerous methodologies were examined. This section examines the strengths and weaknesses of the following classes of computational systems: symbolic systems, connectionist approach, system dynamics, and multi-agent systems. This section is not meant to imply that these are all of the classes of technologies available, not that these classes are mutually exclusive. This section is meant to provide a broad overview of technologies that were examined, and a general comparison as to the strengths and weaknesses of each class.

1. Symbolic Approach – Rule-Based Systems

The symbolic approach builds computational systems using some form of problem solving or planning, and an internal manipulation of symbols based on a logic system such as first-order predicate calculus. An example of a symbolic approach is a rule-based system. A rule-based system typically consists of a start state and goal state,

each of which is represented as a set of facts. The system also contains a set of if-then rules, or productions, and a reasoning, or inference engine. The reasoning engine takes facts from the start state, and attempt to find a premise from a rule that matches these facts. If the reasoning system finds a fact or set of facts that match the premise, it adds the conclusion of the rule to its current fact list. This procedure continues until the goal is discovered or no rules can fire. If the goal is discovered, the set of rules that can be traced from the start to the goal state are a viable solution to the problem. This type of system is derived from Newell and Simon's General Problem Solver using means-ends analysis [Newell and Simon, 1963].

These systems typically solve problems from the top-down; the developer creates a static set of rules prior to run-time that represent possible problem-solving steps. The system iterates through these rules, applies facts, and possibly sub-rules in order to discover a solution.

Rule-based systems have several advantages. First, these systems emphasize the engineering of processes, using a divide-and-conquer approach that may seam intuitive to developers. Additionally, rule-based systems are capable of having an explanation sub-system, which can explain to users, through the rules that fired, how the system arrived at its goal. These systems excel in static, deterministic, perfect information environments.

Rule-based systems have several disadvantages in modeling IA. First, and foremost, in traditional top-down rules-based systems the developer must predefine the rules and actions, and therefore problem solving capabilities, limiting the system to the imagination of developer. In a large system, an engineer may not be able to define all contingencies and combinations *a priori* due to the combinatorial explosion if all combinations were tested [Axelrod, 1997], [Weiss, 1999]. The result is that rule-based systems cannot discover innovative solutions to unforcasted situations.

Additionally, rule-based systems do not deal well in a dynamic, stochastic, or partially observable environments, especially in confrontational domains. In these domains, an entity may be able to discover how a rule-based system acts in a given situation, and therefore adapt to the rule-based system's predictable behavior in order to defeat it.

Simulation systems that are categorized as rule-based include SOAR [Laird *et al.*, 1987] and ACT-R [Anderson, 1993].

2. Connectionist Approach – Artificial Neural Networks

The connectionist approach models cognitive processes based on neural science. If we think of the IA domain as a highly connected, cognitive process, then this approach is a viable computational tool. There are numerous approached to developing connectionist systems, but this section will concentrate on artificial neural nets (ANN).

An artificial neural net consists of a self-organizing, multi-layer network of primitive computational elements, or nodes. The connections among nodes have weights that are adjusted, resulting in the system learning though supervised training. The system is given a limited set of training data, and it adapts its weights to 'fit' the training data. The ANN can be considered a form of statistical inference [White, 1989].

By training an ANN to a set of training data, the system discovers what attributes are important to categorize the training set, in effect, performing pattern matching. If the environment is small, then a large percentage of the situations may be presented to the ANN, resulting in the ANN memorizing the correct categories for all situations. If the environment is very large, then the training set may represent only a very small subset of the possible situations possible, and the system generalizes based on the ANN's implementation and training set.

An advantage to the ANN approach is that the engineer does not have to spend a great deal of time with domain experts trying to enumerate what action to take in every possible situation. The researcher collects a set of real-world examples and uses these as the training set, training the system to generalize on the appropriate action to take. ANNs can make the development of systems feasible where the experts are not available, but historical case studies are available.

Several disadvantages make ANNs inappropriate for modeling IA. First, and foremost, if examples of situations are not in the training set, then they will not be represented internally in the ANN. Second, ANNs learn offline. In a highly dynamic environment, where we wish to model adaptive behavior, ANNs cannot adapt during a simulation execution. ANNs add newly encountered situations and appropriate actions to

their training set and are retrained offline, which can be very time consuming. Additionally, ANNs are very good for generalizing, but may have difficulty with special cases. These special cases may be treated as statistically insignificant noise and be generalized away – eliminating critically important IA events. Additionally, ANNs have no probability distributions on output; a given set of inputs produces a given output, with no probability or explanation of the event occurring. Lastly, ANNs have no way to explain their reasoning. Their answers are derived through the complex interactions of a set of node and link weights, and attempting to understand the ANN's reasoning is very difficult, if not impossible. For these reasons, the connectionist approach was deemed inappropriate.

3. System Dynamics – Stochastic Simulations

In the field of system dynamics, the most appropriate tool for modeling IA is stochastic simulation. Stochastic simulations, or logic sampling, examine empirical evidence in the form of statistical data, and build multi-connected graphs, or belief networks. In the graph, a node represents a state, and a transition between nodes represents a probability of transitioning between states. After building the graph the system generates a large number of models of the domain, by starting at a root in the graph and transitioning to a node representing a significant event. The model results are compared, and the probability of an event occurring is the ratio of the number of times the simulation ended on that node to the total number of simulation runs.

The major advantage of stochastic simulations is that they can provide statistical data on the probability of events occurring. The probabilities in the belief network come from statistical analysis of empirical data. They can also show chains of events that may lead to important events.

A major problem with the use of stochastic simulations in modeling IA is that reliable statistical data may not exist that covers IA at the organizational level. Since this data may not be accurate, the belief network will be flawed, and the results of the simulation will be invalid.

Additionally, the environment of IA is dynamic, with entities adapting to the actions and inactions of other entities. Building a belief network that takes into account

all of the variables and adaptation for a large number of entities presents an intractable problem.

Finally, stochastic simulations break down when researchers are interested in outcomes that occur very rarely. In these cases, the simulation is run a large number of times, discarding noninteresting outcomes. The challenge is that the fraction of interesting, yet rare runs decreases exponentially with the number of evidence variables [Russell and Norvig, 1995]. In any large domain, finding these critical outliers may represent an intractable problem, since the repeated simulations may not discover these statistically insignificant, yet very important situations. This research is interested in those outliers, the statistically insignificant events that may have a catastrophic effect on the security of an organization's information and information systems. For this reason, stochastic simulations were abandoned.

4. Multi-Agent Simulations (MAS)

While there is no commonly accepted definition for agents, Wooldridge proposes a definition that is used throughout this dissertation, "An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives" [Wooldridge and Jennings, 1995]. Other definitions can be found in [Ferber, 1999], [Russell and Norvig, 1995] and [Weiss, 1999].

Multi-agent simulations (MAS) operate from the bottom-up, using multiple adaptive agents "…(as) intelligent actors, interacting among themselves by using their defined attributes and methods, but (are) able to modify those constraints to meet the goals assigned them by the modeler…providing real insight into how best to encourage and take advantage of individual initiatives and adaptability."

Researchers begin by developing the set of actors and objects in the system under investigation, and specify how these interact. Researchers are then able to study the interdependencies between the system components and examine how the system as a whole evolves under varying system parameters. Intelligence emerges through the interaction of many relatively simple autonomous agents [Weiss, 1999].

The socially capable autonomous agents interact within the environment, other objects in the environment, and other agents in an attempt to achieve their own individual goals. The autonomous agents are able to change goals and select actions that they believe can help achieve these goals. A benefit of this bottom-up approach is the ability to integrate new agents and objects into an existing simulation and modify system parameters to perform what-if analysis [Ferber, 1999]. These societies of agents provide researchers insight into the environment under investigation by creating *virtual laboratories* where researchers can explore changes in the agents, environment and society who are modeled after actual (or virtual) components found in actual or fictitious social systems.

MASs have no centralized control -- the agent simulation is leaderless. Each actor (agent) in the simulation independently pursues its own independent goals. Some actors may cooperate while others compete. The result is a highly dynamic environment where software actors, with no human intervention, can search the space of resources and goals and develop innovative solutions for challenges discovered in the environment.

Multi-agent research has been conducted in areas ranging from artificial life [Langton, 1988] to real worlds [Jones *et al.,* 1999]. Varieties of MAS architectures, from purely reactive to cognitive, have been developed to model the various environments.

Cognitive agents (or deliberative agents), from the distributed artificial intelligence (DAI) community, are traditionally based on first-order predicate logic, sophisticated reasoning, and rely on the internal manipulation of symbols. These agents maintain a symbolic representation of the environment within which they operate, and focus on communication and cooperation between agents. Most importantly, these agents have *intentions* -- goals and plans to achieve goals. Cognitive agents inherit the strengths and weaknesses of rule-based systems as discussed above.

Reactive agents, from the field of artificial life (A-Life), are reflexive -- actions are “reactions” to stimulus regulated by perceptions and the agent’s internal state. These agents maintain no planning, history, or symbolic representation of the world. The simple reactive agents are combined into a society, where intelligence is seen as emergent from the vast interactions of the agents and the environment. Refer to Figure 1--

Spectrum of Agent Architectures. See [Weiss, 1999] for a more detailed comparison of cognitive and reactive agents.

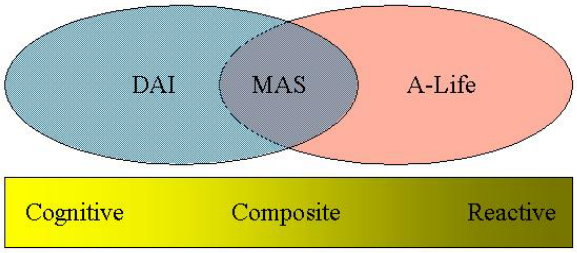


Figure 1. Spectrum of agent architectures.

Unlike cognitive agents, reactive agents do not posses any internal plan or model of the environment. They do not explore alternatives. Rather, they generate actions, and these actions may result in fulfilling their goals. The actions that led to the agent achieving a goal can be studied as an implied plan that emerged from the application of simple reactive rules.

Many properties of reactive multi-agent systems make them a beneficial architecture for the modeling of information assurance. Creating reactive agents is simple when compared to rule-based systems. The systems are computationally tractable and robust. The agents thrive when the environment is stable and tend to adapt rapidly to changes [Axelrod, 1997]. Finally, and most importantly, agents apply simple reactive rules to states and discover "what works". The agent adjusts to the environment, and adapts. This adaptation results in the agent discovering possible new solutions to problems without any explicit plan or engineering bias.

Multi-agent systems are not without challenges. First, reactive agents take a local view, with no long-term plan. This may cause agents to become caught in a 'local maximum', unable to find a more global maximum. There has been little research in the agent's use of exploration of new actions, versus the exploitation of previous successful actions, so agents may discover one path that is successful and abandon searching for additional, better plans. Additionally, MAS systems may have to deal with conflicting goals between agents, which may lead to system gridlock or goal clobbering [Ephrati and Rosenschein, 1994].

From the engineer's point of view, it can be very difficult to 'tune' agents to perform properly in environments. The agent's behavior emerges, and causing the correct behavior to emerge may prove difficult. Finally, the dynamics between many conflicting goals or behaviors, may be very complex and quickly overcome an engineers ability to understand the results of these many interactions [Weiss, 1999].

The purpose of this dissertation is to model the technical and observable social phenomena found in the field of IA. As such, we are not concerned with producing software replicas of actual humans. We wish to create a set of agents who represent individuals found in the IA environment, and whose observable behavior emulates those real-world individuals. To achieve this, a composite agent architecture was created that takes advantage of the strengths of both cognitive and reactive agent architectures [Hiles *et al.*, 2001].

For additional general information on agents and MASs see [Ferber, 1999], [Weiss, 1999]. For seminal MAS architectures and simulations see SugarScape [Axtell and Epstein, 1996], Swarm [Langton, 1997], ISAAC combat simulation [Ilachinski, 1997], and Echo simulated world [Echo, 2000].

E. UNIFIED MODELING LANGUAGE

This dissertation uses the Unified Modeling Language (UML) to formally represent the primitives of the IA model that is introduced. The UML provides a standard graphical notation for visualizing the components in software systems, and for documenting conceptual organizational processes. Developers can use this formal modeling language to express the *structure* and *behavior* of a system [Booch *et al*., 1999].

As stated in [Booch *et al*., 1999], there are four aims in modeling:

- to help visualize a system
- to specify the structure or behavior of a system
- to give a template that guides in construction
- to document decisions made.

UML is used for these purposes. In addition it is "… expressive enough to model nonsoftware systems, such as workflow…. and the design of hardware". The UML is used for two purposes in this dissertation. First, it is used as a graphical notation to illustrate the model developed for this dissertation. The UML formalizes the meaning of the language operators by providing representational rigor and software repeatability. Second, it is used to express the structure and behavior of the software developed as an implementation of the model.

Defining the *things*, *relationships*, and *diagrams* is beyond the scope of this dissertation. A general description of the UML notation used in this dissertation is provided in Appendix B. An excellent user guide is [Booch *et al.*, 1999], and a comprehensive reference manual is [Rumbaugh *et al.*, 1999]

F. SUMMARY

The modeling and simulation of the IA domain has been an ongoing effort for over thirty years. Numerous formal and informal models provide a wealth of information on various portions of the domain, but fail to capture the complex, adaptive nature of IA when it is viewed as a social system.

The next chapter uses lessons learned in multi-agent system design to develop a computational model of IA. The UML is used as a graphical notation to illustrate the model.

III. COMPUTATIONAL MODEL OF INFORMATION ASSURANCE (IA)

A. INTRODUCTION

This chapter introduces the Social-Technical Information Assurance Model (STIAM). STIAM is a computational model of information assurance. The model has two components: a formal model, and a descriptive model. The formal model uses mathematical notation to describe the elements of the model. The descriptive model provides a graphical representation of the formal model. Combined, these two components permit researchers to model elements of information assurance (IA) at the organizational level.

This chapter presents STIAM in formal mathematical notation and depicted graphically using the Unified Modeling Language (UML) [Booch *et al.*, 1999]. Chapter IV introduces a data structure called *connectors*. Connector-based models and notation are introduced, and STIAM is presented in the connector notation.

B. OVERVIEW

The objective of this research is to examine how IA issues affect organizations as a whole. This work does not model individual devices and connections on a network, nor the specifics regarding how information flows through devices and nodes. Rather it focuses on how decisions and omissions made by humans affect an organization at the enterprise level. In order to ask and answer meaningful questions, a precise set of definitions and relationships must be elaborated.

The term *environment* refers to the real-world situation being modeled. In this research, environment is limited to relevant social organizations, people, and information processes and technologies that are responsible for IA issues in the real world. The term *society* represents an abstraction of the environment, depicting generalizations of the entities, their structures, and their relationships within the *environment*.

At the highest level of abstraction, the society contains a group of organizations -- social entities that exist for a particular purpose. From an IA perspective, the components

of the organization are information and people. Information may exist in any format including electronic, paper, punched cards, stone tablets, etc. People use the information to achieve goals. Accessing and modifying information takes place through an organizational infrastructure that contains processes that access and modify the information, and an interface on the infrastructure to allow people to interact with the information. Electronic technology might or might not be embedded in this infrastructure.

From an information-centric perspective, the key components of an organization are:

- Critical information required for the organization to perform its mission.
- Information processes that interact with the information and provide an interface for human actors.
- Key individuals that interact with the information processes directly and thereby interact with the information indirectly.
- Roles that individuals are assigned within the organization that guide them in their goals and provide capabilities needed to achieve these goals.
- The policies and procedures that define the organization.

Information, processes, and roles combine to form organizations, and organizations, infrastructures, and actors combine to form the society. These components are the building blocks of the Social-Technical Information Assurance Model (STIAM) presented in this chapter. The following sections examine these relationships in further detail.

C. SOCIETY

A *society* is comprised of three disjoint domains with each domain containing multiple autonomous entities. The three domains are the *organizations*, *infrastructures*, and people, or *actors*. Formally, a society, *s*, is defined as a tuple relationship expressed in Equation 1:

$$s = \langle O, I, A \rangle$$
where:
O is a set of Organizations
I is a set of Infrastructures
A is a set of Actors

Equation 1. A Society of Organizations, Infrastructures and Actors.

The Organization domain contains a set of abstract representations of social groups. The Infrastructure domain consists of a set of abstract representations of the critical information within an organization as well as the information processes that access and process that information. The Actor domain consists of actors, which are abstract representations of humans critical to IA. Combined, the elements of the domains are the entities in the society. See Figure 2 for a high-level conceptual diagram of the model.

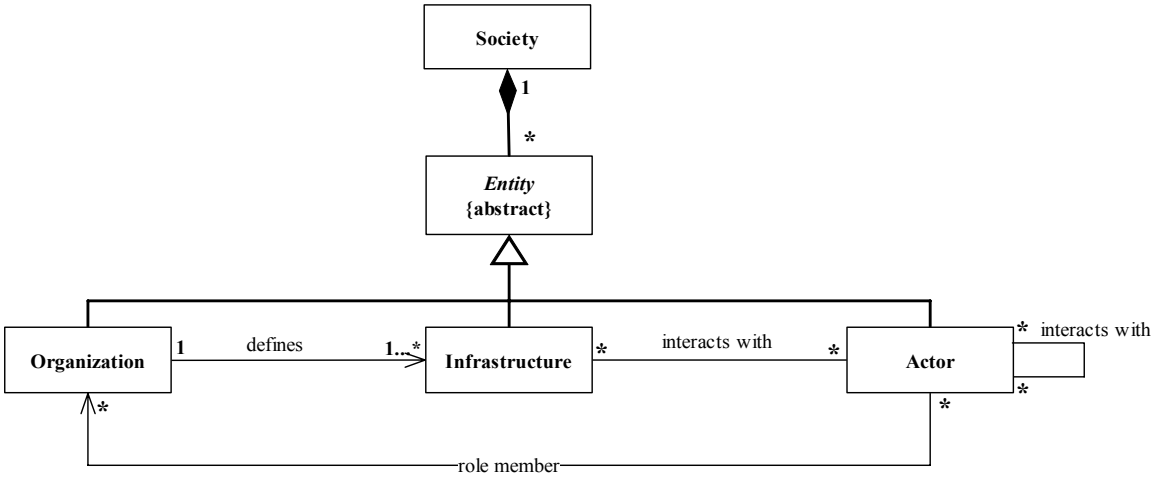


Figure 2. A Conceptual Diagram: A Society composed of Organizations, Infrastructures, and Actors[2].

There are numerous relationships between the three domains of the society. For example:

- a particular Actor may assume multiple roles in multiple Organizations,

[2] This is a conceptual diagram of the IA model. This diagram depicts the major entities, and the relationships between these entities. The components of the entities are not depicted in this diagram to facilitate clarity. See Appendix B for a summary of the UML notation used in this document.

- each Organization may have multiple roles that various Actors can perform,
- a particular Actor may interact with many Infrastructures,
- each Infrastructure may interact with numerous Actors,
- an Organization may define one or more Infrastructures,
- an Infrastructure belongs to a single Organization.

The remainder of this chapter will describe these domains and the associations between them.

D. DOMAINS AND ELEMENTS IN A SOCIETY

Before elaborating on the domains and the elements and associations that make up those domains, a key concept of the model must be introduced, the concept of *tokens*.

1. Tokens

Tokens are an abstract representation of simple static objects that are in the environment being modeled. Tokens allow static objects in the environment to be modeled without significantly increasing the complexity of the model. Tokens may represent passwords, keys, access badges, etc that are found in the environment.

Equation 2 depicts a particular token t_i as an element in the set of all possible tokens T_s in the society s. Figure 3 entitled "The Token Class in UML" depicts a token graphically.

$$t_i \in T_s$$
$$t_i = \langle name_i \rangle$$
where:
T_s *is the union of all tokens in all elements in society s*
$\langle name_i \rangle$ *is a string label for the token* t_i

Equation 2. A Token is an element in the set of all possible Tokens.

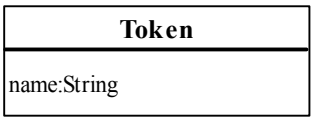


Figure 3. A Token class in UML.

A set $\boldsymbol{T_s}$ consists of all possible tokens in the society s. Tokens are represented in every domain in the model and is discussed as appropriate.

2. Infrastructures

The infrastructure domain ***I*** contains a set of infrastructures. An infrastructure represents the aggregate of the information processing capabilities and the critical information resources found within an organization.

A particular infrastructure $\boldsymbol{i \in I}$ is defined as a tuple as shown in Equation 3:

$$i = \langle IR_i, IN_i, T_i \rangle$$

where:

IR_i *is a set of information resources in infrastructure i*

IN_i *is a set of all interfaces for infrastructure i*

T_i *is the set of all tokens stored on the infrastructure i*

Equation 3. An Infrastructure composed of Information Resources, Interfaces, and Tokens.

An organization possessing information resources must have some means to access and process the information resources. The *infrastructure* represents the information possessing capabilities of an organization in an environment. An interface represents the prerequisite 'handshake' that must occur before actors and other infrastructures are able to interact with these processes.

A set of tokens may be stored on the infrastructure. This represents critical information that is on the infrastructure that may be acquired by actors and utilized to achieve additional infrastructure access. This process is discussed later.

Graphically, the infrastructure relationships are depicted in Figure 4 entitled "An Infrastructure composed of Resources, Interfaces, and Tokens".

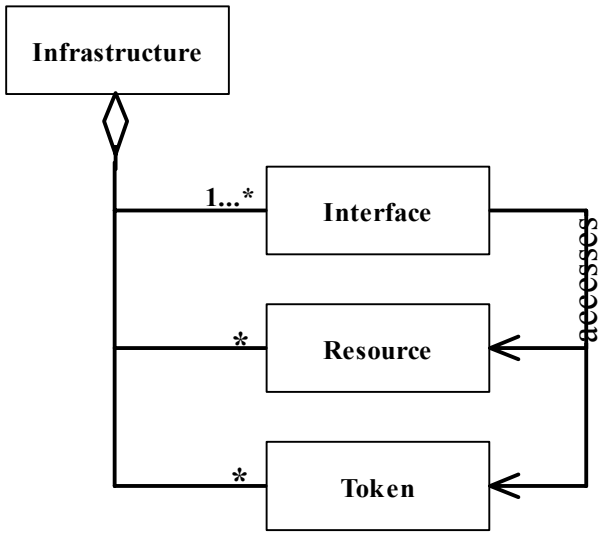


Figure 4. An Infrastructure composed of Resources, Interfaces, and Tokens.

a. Information Resources

Information Resources, hereafter called *Resources*, are the critical information sets whose confidentiality, integrity, and/or availability are required for the organization to exist. Resources represent the content of information and not file objects like data files, email, etc.

Resources will represent different information and processes for different organizations in the environment being modeled. A government organization may be concerned with national intelligence secrets. Financial institutions may be concerned with banking transactions and balances. A particular corporation may be concerned with two resources; one representing sensitive proprietary research and development information, and another representing their customer database. The bottom line is that the resources represent the critical information and processes within the organization, whose disclosure, corruption, or nonavailability may cause harm to the organization or that may be of interest to outside attackers.

b. Interface

Interactions between the infrastructure and other entities occur through an interface. An interface is a mechanism specified by the infrastructure whereby entities that can *connect* with the interface are able to affect the infrastructure and its component

parts. An *interface* contains the requirements necessary for an entity to interact with an infrastructure and a set of actions that occur if those requirements are met. These interactions model information services that employees, customers, or attackers may access to cause effects on the infrastructure and resources.

A particular interface, in_j, is defined by the tuple as shown in Equation 4:

$$in_j = < n_j, s_j, T_j, ae, AC_j >$$
where:
n_j = *name of the interface*
s_j *is the state of the interface where* $s_j \in$ *{active, inactive }*
$T_j \subseteq T_s$ *--a set of interface tokens*
ae is an active entity where $ae \in (A \cup I)$
AC_j = *set of actions*

Equation 4. The components of the interface.

Figure 5, entitled "UML diagram of an Interface" depicts an interface and its components.

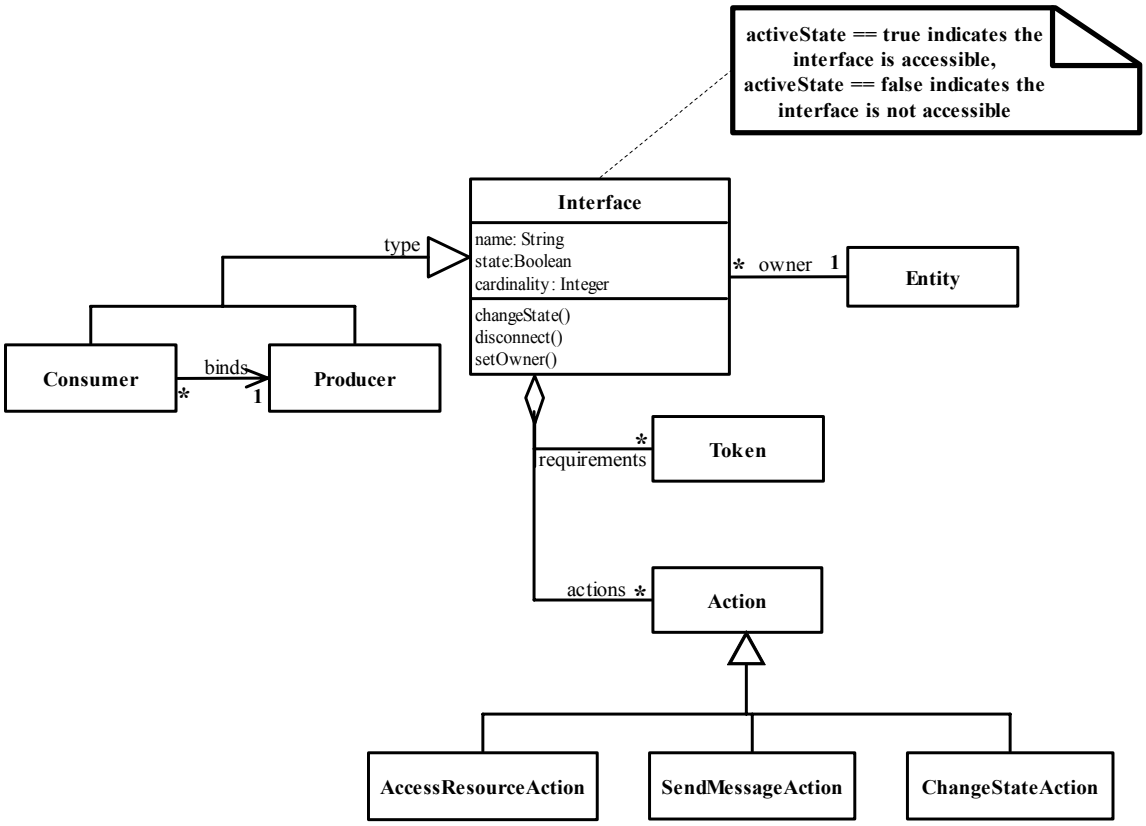


Figure 5. UML diagram of an Interface.

The name of an interface is a simple string label used to refer to the interface. The state property indicates if the interface is accessible at that moment in time. An inactive interface cannot be accessed by another entity. The state property permits a researcher to model the activation and inactivation of processes and services within an infrastructure, permitting the modeling of a dynamic system over time.

Each infrastructure has a set of interfaces that prescribe how actors and other infrastructure may interact with the infrastructure. Additionally, interfaces are used for interacting with actors. The actor interface is discussed later. The interface *owner* provides a pointer to the entity to which the interface provides access.

It is assumed in this model that an interface may permit multiple entities to simultaneously bind at any time. The number of entities that are permitted to bind is specified in the interface's *cardinality* value.

There are two types of interfaces defined: the producer and the consumer. The producer advertises that the infrastructure has a process or action it performs. This may represent a web server, a help desk, database access, etc. The consumer interface advertises that an entity is seeking a matching producer interface; the consumption of the process or action that the producer is advertising. Infrastructures may have any combination of producer or consumer interfaces.

The producer has prerequisites that must be met in order to utilize the services or processes it models. These prerequisites may be something an entity must know (password or phone number), has (a key or physical access to a location), or is (biometric data). These prerequisites are represented as tokens that must be presented by the consumer in order to access the producer interface. If the producer's tokens are a subset of the consumer's tokens then the prerequisites have been met.

The actual mechanism for entities to assume the producer and consumer types, and to interact with one another, is described in detail in Chapter IV. For now, assume an entity that has a goal of consuming a service activates its interface, creates a consumer message, and sends the message to the producer. If the producer's interface is active, the interface names match, and the message contains *at least* the producer's

required tokens, the interfaces are said to *connect*. The connection can be thought of as a temporary contract, where the producer wishes to provide a service, and the consumer wishes to utilize this service. This contract, or connection, is maintained until one of the parties discontinues the connection, at which time the interfaces, and subsequently the owning agents, disconnect.

c. Interface Actions

A set of actions may be designated to execute on the owning entity of the participating consumer and producer interfaces immediately upon a connection occurring, upon a connection breaking, or upon the owning entity's request. These actions permit researchers to model the results of entities interacting in the environment.

There are three types of action results: changing an interface state, accessing a resource, and having an entity send a message to the other party of the connection. Changing an interface state causes an infrastructure to activate or inactivate interfaces, resulting in the addition or removal of capabilities on an infrastructure. This may represent an entity installing, activating, deactivating, or removing services on an infrastructure. Thus, connecting to an interface may result in changing the interface itself.

Some interface actions may result in accessing a resource. These actions represent the actors either *reading* from or *writing* to a resource. This represents a person successfully accessing a critical resource.

An action may send a message to the other party of the interface connection. The message may contain tokens or tickets (tickets are discussed in chapter V). This represents an infrastructure providing additional materials or services to other entities in the binding. This might represent an individual defeating a poor security interface and acquiring the password file, represented as a set of tokens, from a system.

d. Aggregating Infrastructures

An organization may have one or more infrastructures that define the actual capabilities and limitations of that organization's information processes. The decision to maintain individual infrastructures or to aggregate them into a single

infrastructure depends on the goals of the modeler. For example, a researcher may be examining a very large society, with numerous organizations and actors, so he may aggregate the infrastructures in each organization onto a single infrastructure in order to order to observe the 'big picture' and ensure the environment is manageable. On the other hand, a researcher may be interested in more detailed information on a few entities, in which case he may decide to deaggregate the infrastructures in order to model more detail.

3. Organizations

Organizations represent social groups within the environment being modeled. As expressed in Equation 5, an organization $\boldsymbol{o_i \in O}$ is a collection of organizational IA *policies* $\boldsymbol{P_{Oi}}$ and actor *roles* $\boldsymbol{R_{Oi}}$:

$$o_i = < P_{Oi}, R_{Oi} >$$
where:
P_{Oi} *is a set of policies for an organization* o_i
R_{Oi} *is a set of roles for an organization* o_i

Equation 5. An Organization consisting of Roles and Policies.

Organizations do not model actual social groups or collaborations, but are an idealized modeling concept to facilitate insight into IA at the macro level. Organizations range from formal enterprises such as commercial and government entities, to informal collections of individuals with a common goal such as hacker clubs, social groups, etc. The organization may represent a team with heterogeneous, interdependent roles, or a group of homogeneous, interchangeable roles [Kang *et al.*, 1998]. Figure 6, entitled "An organization consisting of Roles and Policies" depicts an organization.

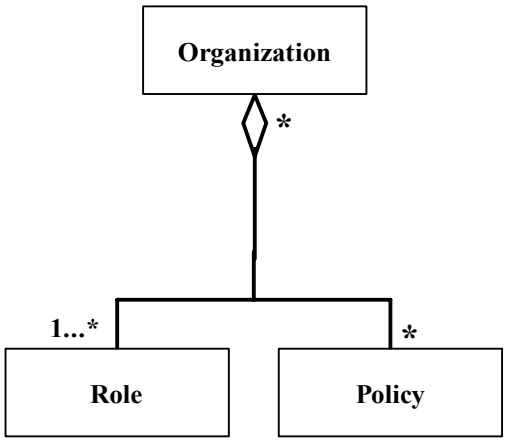


Figure 6. An Organization consisting of Roles and Policies.

a. Policies

Policies are a set of rules *specified* by an organization that state the **desired** restrictions to entities that can access resources, and in what access mode. The term policy, as used here, represent the organization's **desires** to protect the confidentiality, integrity, and availability of an organization's information resources, not the actual implementation. The implemented technical security policy [Brinkley and Schell, 1994] is implied in the interfaces and their subsequent actions on the infrastructure.

Policies are *not* access validation rules, such as access control lists or capability lists, used to determine access decisions on the infrastructures. The policies are the *desired and specified* restrictions to resources, what Sterne calls the Security Policy Objectives; "A statement of intent to protect an identified resource from unauthorized use…" [Sterne, 1991]. The security of an organization can "only be said to be 'secure' with regard to some specific security policy, stated in terms of controlling access of persons to information" [Brinkley and Schell, 1994]. This being the case, the policy set provides a means for detecting a violation of the organization's information security.

A policy $p_k \in P_{Oi}$ is declared by the tuple:

$$p_k = \langle e, ir_k, m, au \rangle$$
$$\text{where:}$$
$$e \in (A \cup I \cup \{*\})$$
$$\text{where } * = \text{entity wildcard character}$$
$$ir_k \in IR_i \text{ is the resource the policy refers}$$
$$m \in \{read, write\} = \text{access mode}$$
$$au \in \{permit, forbid\} = \text{authorization}$$

Equation 6. A Policy consisting of an Entity, Infrastructure, Mode, and Authorization.

If a particular entity, *e* referred to in the security community as the *subject*, accessed some resource ir_k in some mode *m* that is not permitted, or that is explicitly prohibited by a policy, then that entity has violated the organizational security policy.

The subject of the policy, *e*, was deliberately chosen to be an entity, rather than an actor. This is because, an actor *a* may connect to an infrastructure interface in_1, which may cause an action to execute. This action, executing on ir_1, may cause ir_1 to attempt to connect to another interface on another infrastructure, in_2. In this case, the active entity making the connection to in_2 is an infrastructure, not an actor.

Read and *write* modes are all that are required to describe rules for access [Brinkley and Schell, 1994]. Connecting to an *interface* may cause the *execution* of a particular action, but this action subsequently needs to access a resource in *read* or *write* mode to cause any significant IA event. Therefore, an explicit execution mode is not required for this model.

The policy mechanism as described above is very robust, permitting the expression of both open and closed security policies [Lunt, 1989] using the *entity wildcard character* '*' literal. This character represents the union of the actor and infrastructure sets, and allows the expression of all active entities as being permitted or forbidden access to a specific resource.

A closed policy forbids all accesses except those that are explicitly permitted. An open policy permits access to all resources that are not explicitly

forbidden. To express the former, the researcher adds policy rules forbidding all entities access to a resource ir_k for both access modes:

- <*, ir_k, read, forbid>
- <*, ir_k, write, forbid>

Next, the researcher selectively adds rules to record permitted access to resource ir_k to selective entities. To express an open policy the researcher explicitly permits all access, and then add rules to selectively forbid access to certain entities.

The policies as specified create two partitioned sets of policies, those that are permitted and those that are forbidden. These sets are independent, and may conflict [Lunt, 1988], resulting in permitted and forbidden authorization simultaneously. These cases may represent actual situations in the environment, and must be reconciled if they occur. There are numerous subtleties in expressing and implementing policies, as addressed in [Lunt, 1988], and [Brinkley and Schell, 1994].

Finally, the actual implementation of security policies are embedded in the infrastructures. The infrastructure interfaces and actions should support the organizational policies, but in reality, there might exist a means to bypass these policies on the infrastructures. An entity may bind with an infrastructure using the system capabilities in a way that is possible, but that violates the policies of the organization. The policy system as specified provides a means to detect these violations.

b. Roles

The set of actor roles $\boldsymbol{R_{Oi}}$ is a collection of defined behaviors specified for an organization. These roles are placeholders, initially defined but unfilled by actors. Roles are discussed in the following section.

4. Organizational Roles

A role is a relationship between an organization and an actor. A role can be thought of as a placeholder within the organization that an actor may fit. The role directs specific actions on the participating actor by providing *goals* to the agent to pursue. Some of the typical roles critical to an IA simulation are system users, system administrators, managers, cyber attackers, and vendors.

It is assumed in this model that all goals and actions are derived from the actor's assigned roles. Furthermore, it is assumed that an actor commits to roles, and that roles are voluntary.

A particular actor a_k may commit to a particular role $\boldsymbol{r_i \in R_{Oj}}$. The role is depicted in Figure 7 and is defined by the tuple:

$$r_i = < RG_{i,}, RQ_i, T_i>$$
where:
RG_i is a set of Role Goals provided by the role
RQ_i is a set of prerequisite Role Requirements
T_i is a set of Tokens provided by the role

Equation 7. A Role consists of Role Requirements, Role Goals, and Tokens.

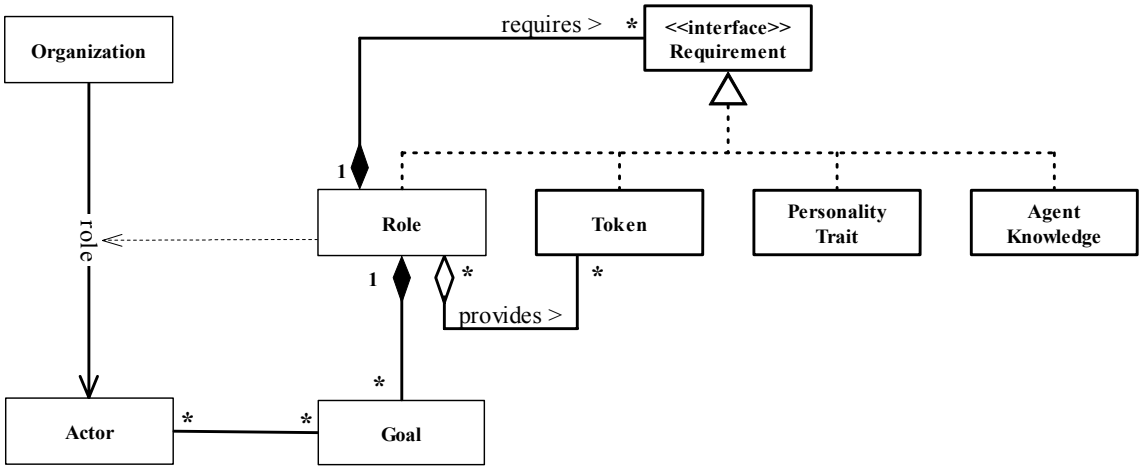


Figure 7. A Role and its components.

a. Role Goals

Role Goals, RG, are desires an agent pursues. Actors who commit to a role are given goals that are then added to the actor's goal set *G*. These goals represent additional commitments that the agent must pursue. Goals have priorities, and new, higher priority goals may eliminate older lower priority goals, thereby causing the impression of goal elimination. Actors assign priorities, or weights, to goals to aid in resolving goal role conflicts. Actor goals are discussed in detail in Chapter V.

b. Role Requirements

Roles have requirements that must be met prior to assuming a role. It is assumed that any actor that fulfills these prerequisites is permitted to assume an

unoccupied role. These prerequisites may be tokens, prerequisite roles, or some particular actor knowledge or personality attribute (see Chapter V for a discussion of personality and skill sets). Roles may also have corequisites that must be maintained. Failure to maintain corequisites may result in the role being revoked by the organization such as being fired or thrown out of a group.

A role requirement RQ_i is defined formally by the tuple:

$$RQ_i = < R_i, T_i, BM_i, K_i >$$

where:

R_i *is a set of prerequisite Roles*

T_i *is a set of prerequisite Tokens*

BM_i *is a set of personality traits, referred to as Behavior Moderators*

K_i *is a set of agent skills or knowledge*

Equation 8. Role Requirements as a Collection of Sets.

The role requirement is a collection of sets as defined in Equation 8. An actor requesting to assume the role must possess each of the elements in each set. Roles are discussed in detail in Chapter V.

c. Role Cardinality

Role cardinality refers to the number of actors who may simultaneously fill a single role. Role cardinality is always one. If an organization contains multiple homogeneous roles, then they are represented as duplicate roles to which different actors may commit. Figure 8, entitled "Multiple homogeneous System Administrator roles," depicts two homogeneous roles, both being system administrators. Each role is individually instantiated, and actors can commit to them individually.

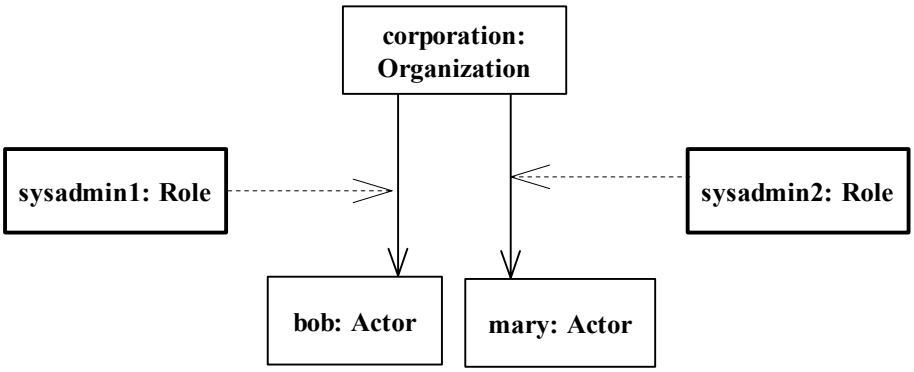


Figure 8. Multiple homogeneous System Administrator roles.

d. Role Tokens

Tokens provided as a component of a role represent objects provided by an organization to a role member to assist in performing the role. Tokens may represent authority required, authorization tokens, or other physical or logical objects or properties in the environment being modeled. Access tokens may be:

- physical, such as access to a location,
- logical, such as read and write permissions on data objects,
- social, such as the ability to interact with other actors.

Tokens may include other objects provided by the organization including financial assets, software, devices, physical tools, etc. In some cases an organization may not provide all of the tokens required to satisfy the goals of a role, so an actor may need to use some other means to obtain the tokens to satisfy the goal, such as acquiring them from other actors or infrastructures. This is discussed in Chapter VI.

5. Actors

An actor is an abstract software representation of IA-relevant humans in an environment being investigated. Actors may represent individual humans, such as each member of a very diverse research team, or may be an aggregate of a relatively homogeneous set of individuals, such as all of an office's cleaning staff. If a researcher is modeling aggregates of individuals, then the individual roles must represent aggregated roles.

Actors are "cognitively limited" [Prietula, 1998]. Since this research is not interested in building complex cognitive representations of the human thought process, relatively simple reactive software agents can be used.

An actor receives its capabilities and desires from the roles to which it is committed. The roles may provide tokens to aid in the satisfaction of a goal. Additionally, a role provides a set of one or more goals that are commitments to the Roles. Goals need procedural knowledge so that the agent can pursue and achieve these goals, and an action set, which are the capabilities that the actor can execute.

Figure 9 illustrates conceptually the major components of an actor. The diagram depicts an actor defined in terms of the roles to which it has been committed. The roles provide the actor with goals it attempts to fulfill. The procedural knowledge provides the actor with the means to achieve the goals, utilizing the actor's available tokens. Finally, the actor selects actions to perform. These actions attempt to interact with other entities through their interfaces. The actor also possesses interfaces that permit other entities to interact with the actor. Successfully connecting to an actor's interface may result in the actor executing another action, possibly changing that actor's internal components or goal priorities. The actual agent implementation is dependent on the researcher's goals. An implementation of an agent that contains these capabilities is provided in Chapter V.

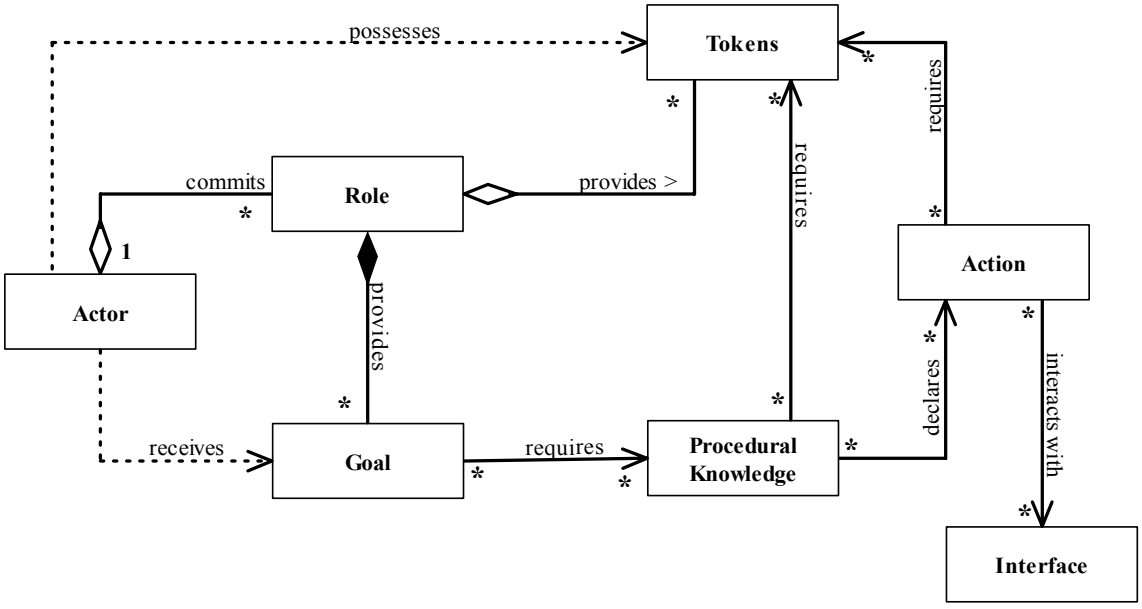


Figure 9. Conceptual diagram of an *actor* entity.

E. SUMMARY

This chapter introduces the Social-Technical Information Assurance Model (STIAM). The formal and descriptive model permits researchers to investigate a society of organizations, actors, and infrastructures, and their component elements at the organizational level. This model permits researchers to model an environment at a variety of abstraction level. STIAM is designed to be implemented as a computational software system to permit researchers to investigate the society and its elements and their interactions.

Chapter IV will translate STIAM into a multi-agent model. This is performed using a data structure called iconnectors. The connector-based model of IA uses a special graphical notation that is useful for clearly modeling a society, and visualizing the dynamics of the system of elements. Chapter V provides a detailed description of a software agent architecture that was developed to implement the actors in this model.

IV. CONNECTORS AND A CONNECTOR-BASED MODEL OF INFORMATION ASSURANCE

A. INTRODUCTION

This chapter introduces *connectors*, a powerful communications mechanism for developing computational models of complex domains, and for implementing these models in software simulation systems. This chapter builds on the mechanisms as proposed by John Hiles [Hiles *et al.*, 2001] and discussed in [VanPutte *et al.*, 2001], [Hiles *et al.*, 2002], and [Osborn, 2002].

This dissertation develops and implements two types of connector mechanisms. The first, simply called '*connectors*', was proposed by John Hiles [Hiles *et al., 2001*] and first implemented by Brian Osborn as part of his dissertation [Osborn, 2002]. This connector mechanism was extended and used strictly as an internal communications mechanism within actors, as discussed in Chapter V. The second type of connector, called infrastructure connector or '*iconnector'*, is used in this research as an inter-entity communications mechanism. This chapter makes extensive use of the iconnector mechanism.

This chapter presents the concept of iconnectors, including a formalism for defining iconnectors and their interactions. It then provides a graphical notation for illustrating connector-based models. Finally, it describes the components of iconnectors in detail, and relates these components to the computational IA model presented in Chapter III.

B. ICONNECTORS

1. Introduction

Computational systems that contain numerous autonomous entities require a mechanism to facilitate communication between entities. Numerous communications mechanisms have been proposed and implemented in multi-agent systems. See [Weiss, 1999] for a summary of agent communication mechanisms and protocols.

For purely communicating agents [Ferber, 1999] such as those proposed in the IA model, we propose a lightweight communication mechanism called *iconnectors*.

2. Definitions

Iconnectors are a lightweight inter-agent communications mechanism inspired by biology. Iconnectors resemble the mechanisms that support chemical flow through multi-cellular membranes[3]. The cells of a multi-cellular organism communicate using proteins that extend through the cell membrane walls. These proteins sense a cell's outer environment and allow passage of materials in and out of the cell. Carbohydrate chains are attached to the proteins on the outer layer. A molecule external to the cell that matches the carbohydrate tags cause a "signal" to travel through the carbohydrate into the protein, thereby signaling a change in the cell.

The result of building iconnector-enabled agents is a biologically based computational system. Iconnectors use a process called connecting [Hiles *et al.*, 2002] or *binding* to facilitate inter-agent communications. Communications between entities occur via this binding, first during setup, and possibly later when using the link.

Entities create iconnector objects that represent a desire to communicate with other entities. These iconnectors are registered with a singleton entity called *ibinder*. The ibinder acts as the switchboard, attempting to find another iconnector that *matches* the first. If the ibinder finds a match it registers these two iconnectors as connected, and they are said to *bind*. It is at this point that the entities are said to be communicating through the iconnectors. Thus, the iconnectors act as facilitators of communications, and the ibinder acts as the digital switchboard binding iconnectors that match.

Figure 10 depicts the three major components in the iconnector process.

[3] Based on a personal conversation with John Hiles, Naval Postgraduate School, Monterey, California, June 2001.

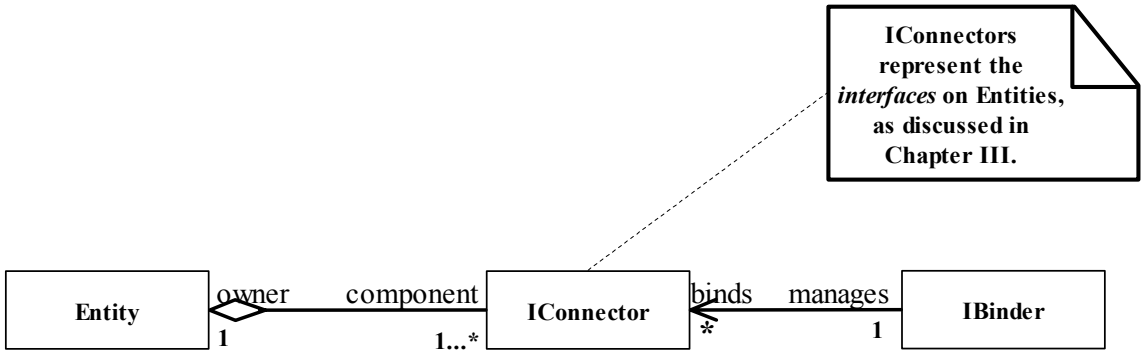


Figure 10. The IBinder *binds* Iconnectors, which are components of Entities.

3. Iconnectors and Entity Interfaces

Iconnectors are used in the IA computational model to implement communications between entities. The iconnectors are a simple mechanism for implementing the functional specifications, connection setup, and communications *interfaces* for the numerous entities in the model.

An iconnector is an "active object that can sense and react to the environment" [Hiles *et al.*, 2001]. An iconnector is not a passive property of an entity. Iconnectors react to operations performed by the owning entities and have the potential for affecting other entities. These effects are realized through sets of *actions* that can be associated with iconnectors that execute on the owning entity upon the iconnector changing state and bindings.

4. Formalism

An Iconnector is defined in Equation 9 and depicted graphically in UML in Figure 11, entitled "Iconnector Class Diagram".

An Iconnector c_i is defined by the tuple:

$$c_i=<l_i, e_i, s_i, ca_i, AC_i, ty \mid l_i \in L, e_i \in (A \cup I), s_i \in S, ca_i \in I^+, ty \in TY>$$

where:

$L = \{n, T_i \mid n = \text{name}, T_i \subseteq T_s\}$ = label

$e \in (A \cup I)$ = an active entity in the society s

$S = \{extended, retracted\}$ = *set of iconnector states*

$I^+ = \{0,1,2,3,...\}$ = *cardinality*

AC_i = *set of actions*

$TY = \{socket, plug\}$=*set of iconnector end types*

Equation 9. An iconnector specification consisting of Labels, State, and Cardinality.

Equation 9 states that an iconnector is composed of a label, owning entity, state, cardinality, set of *actions,* and *type* designation. A label l_i expresses the requirements needed to bind to an iconnector; a name for the iconnector, and tokens required to interact with the interface. The researcher determines appropriate names for all interfaces on all entities. These names are used to designate desired communications with other iconnectors.

Each iconnector has an entity pointer that indicates the entity for which the iconnector is an interface.

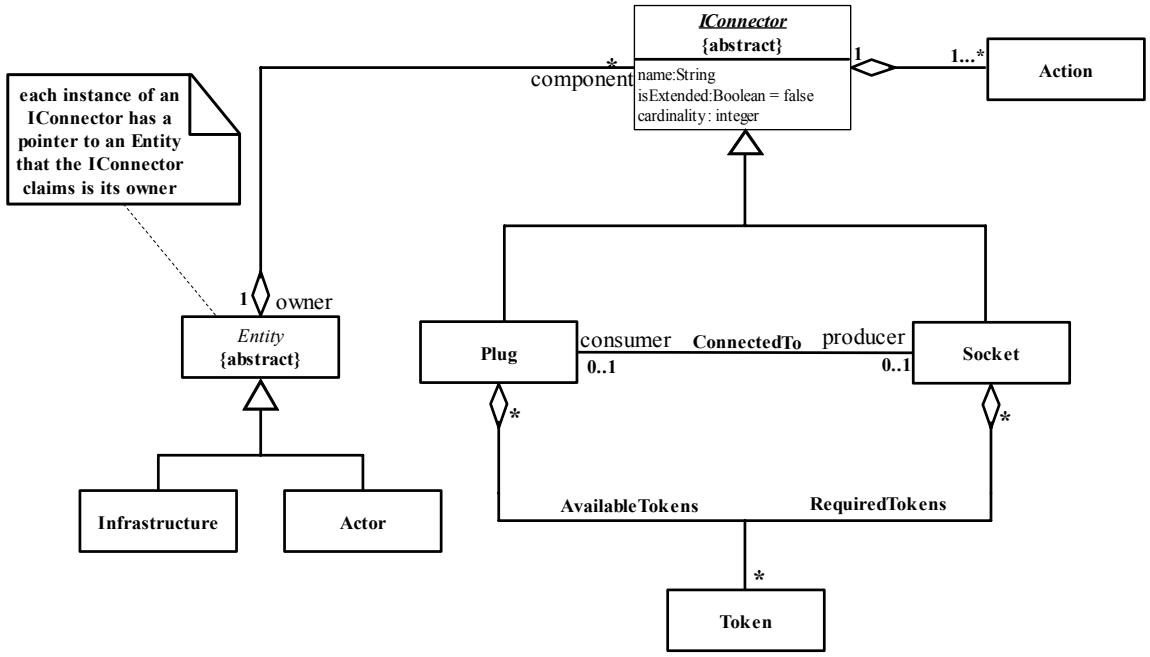


Figure 11. Iconnector Class Diagram.

Each of the components of the iconnector are discussed in detail in the following sections. Before presenting all of the components of iconnectors, a simple iconnector notation is presented.

C. ICONNECTOR GRAPHICAL NOTATION

The iconnector notation is used to depict a connector-based system. The advantages of the iconnector-based notation are brevity and clarity. The notation allows a security analysts to depict entities and their relationships though simple diagrams, and this notation permits security analysts to visualize the society as the states of the entity interfaces change over time. UML diagrams can be used to append additional detail as necessary.

In the iconnector notation, entities are depicted as in Figure 12. Actors are circles, infrastructures are rounded rectangles, resources are triangles within the owning infrastructure, and organizations are octagon. Name labels are provided above the components. Goals, roles, and organizations are not depicted in the basic iconnector notation, but are discussed in Chapter V.

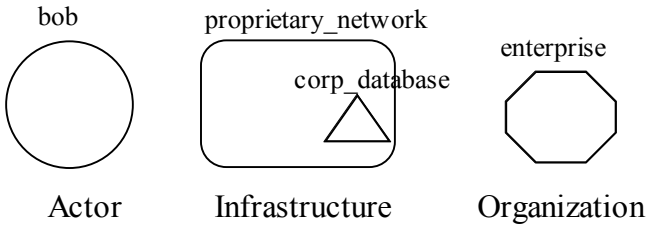


Figure 12. An Actor 'bob', an Infrastructure 'proprietary_network with a Resource 'corp_database' and an Organization 'enterprise'.

A simple iconnector diagram is depicted in Figure 13. This diagram depicts an actor 'bob', with a plug iconnector extended and an infrastructure 'enterprise' with a socket iconnector extended. Graphically, an iconnector is rendered as a solid line, beginning within an 'owning' entity, and directed out of the entity, with an end symbol depicting the type of iconnector. These iconnectors represent an implementation of the

interfaces as depicted in Chapter III. The details of the iconnectors are in the following section.

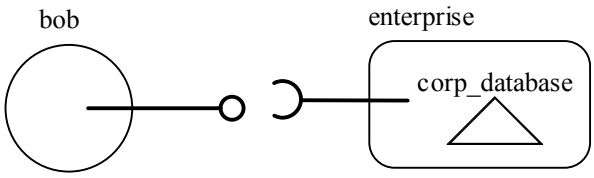


Figure 13. Entities with Iconnectors added.

The mediating ibinder object is not depicted on the diagram. The iconnectors are drawn as if they extend and retract in the society, but in implementation, the ibinder handles the mechanism of the binding process. Elimination of the ibinder from the diagram improved the clarity of the drawing.

The iconnector label may be depicted on the iconnector diagram.

D. ICONNECTOR COMPONENTS

An iconnector has several properties and functionalities. A detailed discussion of the components is provided in this section.

1. Iconnector State – Extended or Retracted

Iconnectors have a Boolean state: extended or retracted. A retracted iconnector is inactive, and cannot connect to any other iconnector. An extended iconnector is currently available for connecting. Extending and retracting iconnectors is a symbolic way of saying that functionality is enabled or disabled. If a connection occurs, and one of the iconnectors subsequently retracts, then the binding is broken, and subsequently the remaining extended iconnector may connect to another extended iconnector. An extended iconnector is distinguished from a retracted iconnector graphically by a small perpendicular tick on the retraced iconnector as depicted in Figure 14.

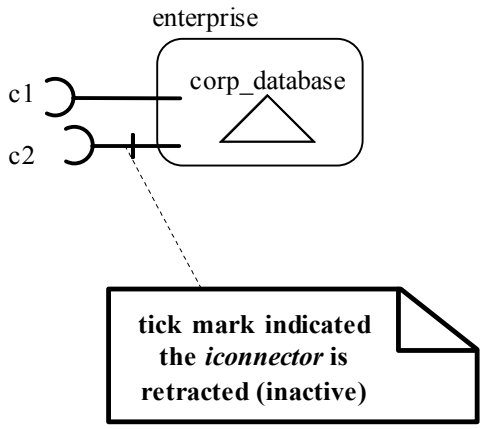


Figure 14. Extended and Retracted Iconnectors for an Infrastructure.

Infrastructures have iconnectors that represent interfaces to processes on an organizational infrastructure. Binding to one of these iconnectors may cause actions to execute that cause other iconnectors to extend or retract representing the activation or inactivation of processes on the infrastructure. Actors extend and retract iconnectors, which represent actions the actor is performing in the support of goals.

Iconnectors are extended and retracted to and from both actors and infrastructures in order to advertise or request access to either resources or actions. When an entity advertises that it has a capability, the diagram notation indicates that it extends a socket iconnector. When an entity requests a resource, the diagram notation indicates that it extends a plug iconnector. If a socket accepts a plug, the two iconnectors are said to bind and are drawn as such.

Iconnectors can extend without the owner of the connection being aware of this event. A 'hidden' iconnector might represent functionality on an infrastructure that is not an advertised capability. For example, a buffer-overflow vulnerability on a server might be represented as a 'hidden' socket iconnector, with required tokens that represent knowledge of the vulnerability and skills required to exploit the vulnerability. Requirements to bind are always represented as tokens and are discussed below.

2. Iconnector Types – Sockets and Plugs

There are two types of iconnector ends: sockets and plugs. Sockets represent interfaces to access resources and actions– a means to achieve goals. When an agent

requires a resource or action, it extends a plug iconnector and requests to bind to a socket. If a socket exists that matches the plug parameters then the requesting agent *binds* to the resource or action. The sockets and plugs match the producer and consumer types on interfaces respectively, as discussed in Chapter III.

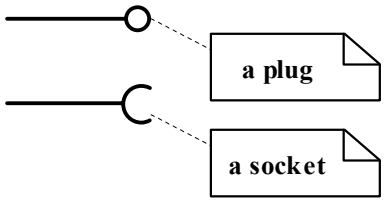


Figure 15. Socket and Plugs depicted graphically.

Socket labels differ from plug labels only in their use of tokens. The tokens listed on a socket (T_{socket}) are the required tokens that *must* be presented to bind to this socket. The tokens listed on a plug (T_{plug}) are the tokens available to the owner of the plug.

Binding to a socket simulates the plug owner's desire to access the requested resource. The initiator must posses all of the required tokens in order to make this binding ($T_{socket} \subseteq T_{plug}$). Equation 10 depicts the requirements to bind mathematically.

$$
\begin{array}{c}
\textit{Given:} \\
\textit{plug } c_p = <l_p, e_p, s_p, ca_p, AC_p, t_p \,|\, l_p = <n_p, TP_p>> \\
\textit{socket } c_s = <l_s, e_s, s_s, ca_s, AC_p\, t_p \,|\, ls = <n_s, TP_s>>, \\
\textit{a binding occurs iff:} \\
n_p = n_s \quad \textit{//names must match} \\
T_p \subseteq T_s \\
s_p = s_s = \textit{extended} \\
ca_p > 0,\ ca_s > 0 \\
\textit{and neither is currently bound} \\
t_p = \textit{plug}, t_s = \textit{socket}
\end{array}
$$

Equation 10. The binding of a Socket to a Plug iconnector.

3. Iconnector Cardinality

Iconnectors have a cardinality that specifies the number of iconnectors that can simultaneously be bound to this particular iconnector. An iconnector without a cardinality label has a default cardinality of one. An iconnector with a cardinality of zero represents a special type of iconnector called a Listening iconnector (see Section IV.D.5). See Figure 16 for examples of iconnector cardinality labels.

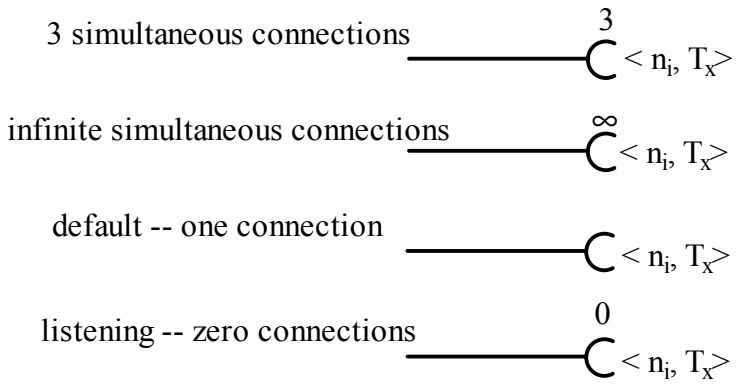


Figure 16. Socket cardinality diagramming convention.

4. Iconnector Labels

Iconnectors are depicted with one end as a socket or plug iconnector and the other end intersecting one edge of the owning entity. See Figure 17 for a depiction of an actor's plug iconnector attempting to bind to an infrastructure's socket.

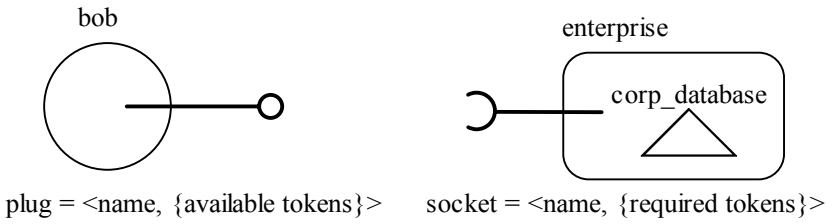


Figure 17. An Actor Plug attempting to bind to an Infrastructure Socket Iconnector.

If a plug iconnector with all of the prerequisite tokens is presented to a socket iconnector, then they bind and *actions* may result.

5. Listening Iconnector

A listening iconnector is used in situations where an infrastructure or actor wants to be notified that a iconnector exists in the society, but does not wish to bind to this iconnector. This may occur, for example, when an agent wants to know if a particular resource exists and is available on an infrastructure. If a listener iconnector discovers the requested iconnector, the owning actor is notified of the iconnector on the entity, without the entity being notified of the actor's query.

Listener iconnectors can be distinguished from other iconnectors by the cardinality label of zero. Figure 18 (a) depicts a listener plug iconnector that notifies the actor if a socket becomes available that matches the listener's label. Figure 18 (b) depicts an infrastructure that will extend a iconnector c_y if an entity extends a plug that matches the socket iconnector c_x. The mediating ibinder (not shown) takes care of the underlying listening iconnector mechanism.

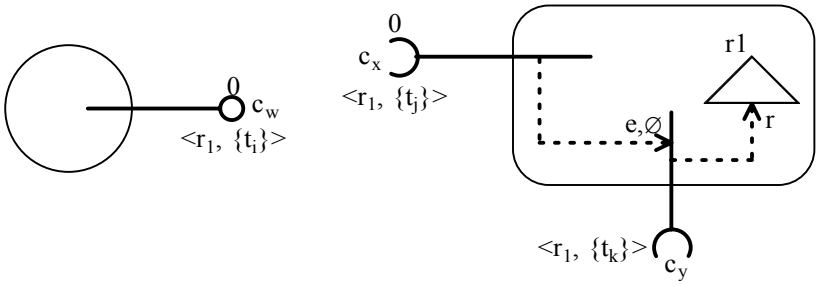


Figure 18. Listening Iconnectors.

5. Actions

Actions represent the effects of successfully accessing an infrastructure or actor through an interface. There are three types of actions:

- resource action
- connection action
- message transfer

Resource actions represent the access of a critical information resource. Connection actions represent a modification to the interface of an infrastructure. Message transfers represent an entity providing potentially additional capabilities to other entities. All actions are depicted graphically as dashed, directed lines, called *action lines*.

a. Resource Action

Resource actions are depicted graphically as dashed lines that intersect an owning iconnector inside an owning entity with an arrowhead on the other end of the action line pointing to a resource inside the infrastructure. The resource action line has a label l_{ra}:

$$l_{ra} = <mode|mode \in \{r,w\}>$$
where r = read access and w = write access

Equation 11. A Resource Action definition.

When an entity binds to an iconnector that has a resource action attached, the iconnector is now accessing the resource in the mode indicated by the action label. When the entity disconnects from the iconnector it is no longer accessing the resource. This action represents an entity accessing information resources.

b. Connection Action

Connection actions are depicted graphically as dashed lines that intersect an iconnector inside an owning entity with the arrowhead on the other end of the action line pointing to the iconnector that is affected by the binding. The connection action line has a label l_{ca}:

$$l_{ca} = <\mathbf{s_1}, \mathbf{s_2}> where\ \mathbf{s_1}, \mathbf{s_2} \in \{\varnothing, e, r\}$$
where ∅ = no action, e= extend, r = retract

Equation 12. A Connection Action definition.

A label is placed by the arrowhead with the double $<s_1, s_2>$. The value of s_1 is the effect ***to the iconnector pointed to***, when the owning iconnector becomes bound. Likewise, s_2 is the effect on disconnecting, where the effect is null, extends, or retracts respectively. An iconnector that is given the action to extend or retract, and is already extended or retracted respectively, will not perform any action. The action connection represents a modification to an infrastructure interface that can occur when an entity successfully connects to the infrastructure.

Figure 19(a) depicts a condition where an iconnector c_y is by default retracted, and therefore the resource r_1 is not accessible. If a successful binding occurs to iconnector c_x, then the connection action on c_x will execute, causing c_y to extend. This state is depicted in Figure 19(b). If an entity binds to c_y at this point, c_y will execute its resource action resulting in read access to resource r_1. On disconnecting from c_x, the iconnector c_y will retract and any entity connected to c_y will be forced to disconnect.

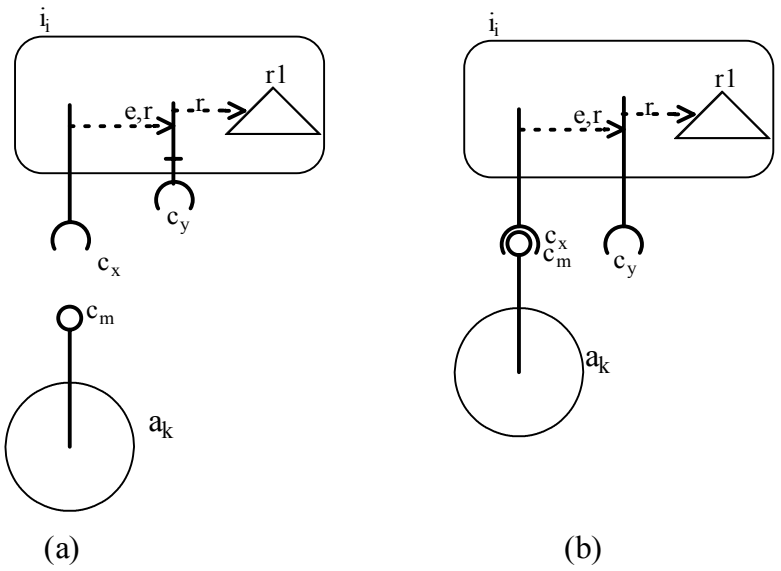


Figure 19. Actor a_k binding to an Iconnector and a different Iconnector reacting by *extending.*

Multiple iconnectors can be combined to react to each other as in Figure 20. This diagram depicts a condition where, if either socket iconnector is bound, the other retracts. This example depicts a binding to resource r_i if token t_i ***or*** t_j is presented. Upon binding to one of the sockets the other socket retracts through the iconnector action command. In this example, if an entity extended a plug with t_i ***and*** t_j tokens, the plug can bind to either one of the sockets (a stochastic choice make by the ibinder), and the other socket then retracts.

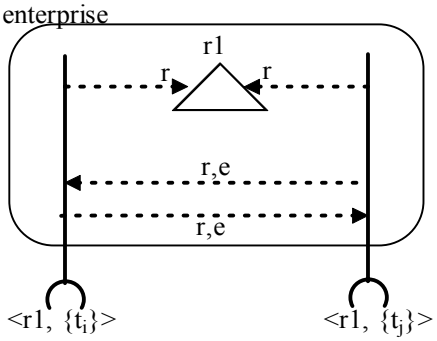


Figure 20. Connectors that permit Token t_i or t_j.

c. Message Transfer

In some situations, an iconnector binding causes an entity to transfer an object to the other party of the binding. The transfer message, hereafter referred to as a *message*, facilitates the exchange of tokens and capabilities between entities. A transfer message is depicted graphically as a 'lightning bolt' arrow, with the base of the arrow touching a token or ticket that is transferred, and the head of the arrow pointing toward the destination entity of the message.

A transfer message is a one-way, directed communication between entities. A source entity specifies the destination entity and the contents of the message, consisting of tokens and/or tickets (discussed in Chapter V).

Figure 21 (a) depicts an example of a message from an infrastructure. If an entity binds to socket c_1, an action causes the message containing ticket tk_1 to be sent to the entity that bound to c_1. Figure 21(b) is similar. If an entity binds to c_2, the actor sends the token T1 to the binding entity.

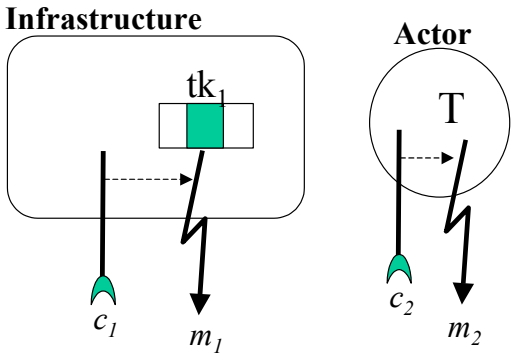


Figure 21. An Infrastructure and an Actor with *transfer* messages.

Messages convert the owner of a message into a vendor, who can transfer tokens and tickets to other entities. The receiver of the message can choose to accept this message, in which case the receiver now possesses this new capability, or ignore the message.

E. SOCKET AND PLUG CONNECTIONS

The socket and plug mechanism can best be explained by an example.

Figure 22 is a sequence diagram that depicts the interactions that take place between two entities that extend iconnectors to the ibinder, and then later one of the iconnectors retracts. The diagram depicts the following high-level events:

1. entity1, an Entity creates a Socket, and calls its `extendConnector` method() which changes the Socket's state to *extended*.
2. The Socket notifies the IBinder that its state has changed, and the IBinder registers the Socket as an active socket connector.
3. entity2 creates a Plug, and calls it's `extendConnector` method() which changes the Plug's state to *extended*.
4. The Plug notifies the IBinder that it's state has changed, and the IBinder registers that Socket as an active plug connector. Next, the IBinder checks if the two sockets are compatible, which they are.
    1. The IBinder calls the Socket's `connect`() method, notifying the Socket that it has connected to Plug.
    2. The Socket calls all of its actions that execute once a connection has been made.
    3. The IBinder calls the Plug's `connect`() method, notifying the Plug that it has connected to the Socket.
    4. The Plug calls all of its actions that execute once a connection has been made.
5. At this the point two entities are bound through their respective iconnectors. These entities may now communicate directly.
6. Next after some activity, entity1 decides that it no longer needs to be connected, and calls the Socket's `retractConnector`() method.
7. The Socket changes it's state to *retracted*, and notifies the IBinder.
8. The IBinder removes the Socket as an active iconnector and notifies the Iconnector that the Socket is no longer connected, by calling the Socket's `disconnect`() method.
    - The Socket calls all of its actions that execute when a connection is broken.

9. The IBinder notifies the Plug that the Plug is no longer connected, by calling the Plug's `disconnect`() method.
    - The Plug calls all of its actions that execute when a connection is broken.
10. The IBinder attempts to reconnect the Plug to an existing the Socket whose label matches the Plug.

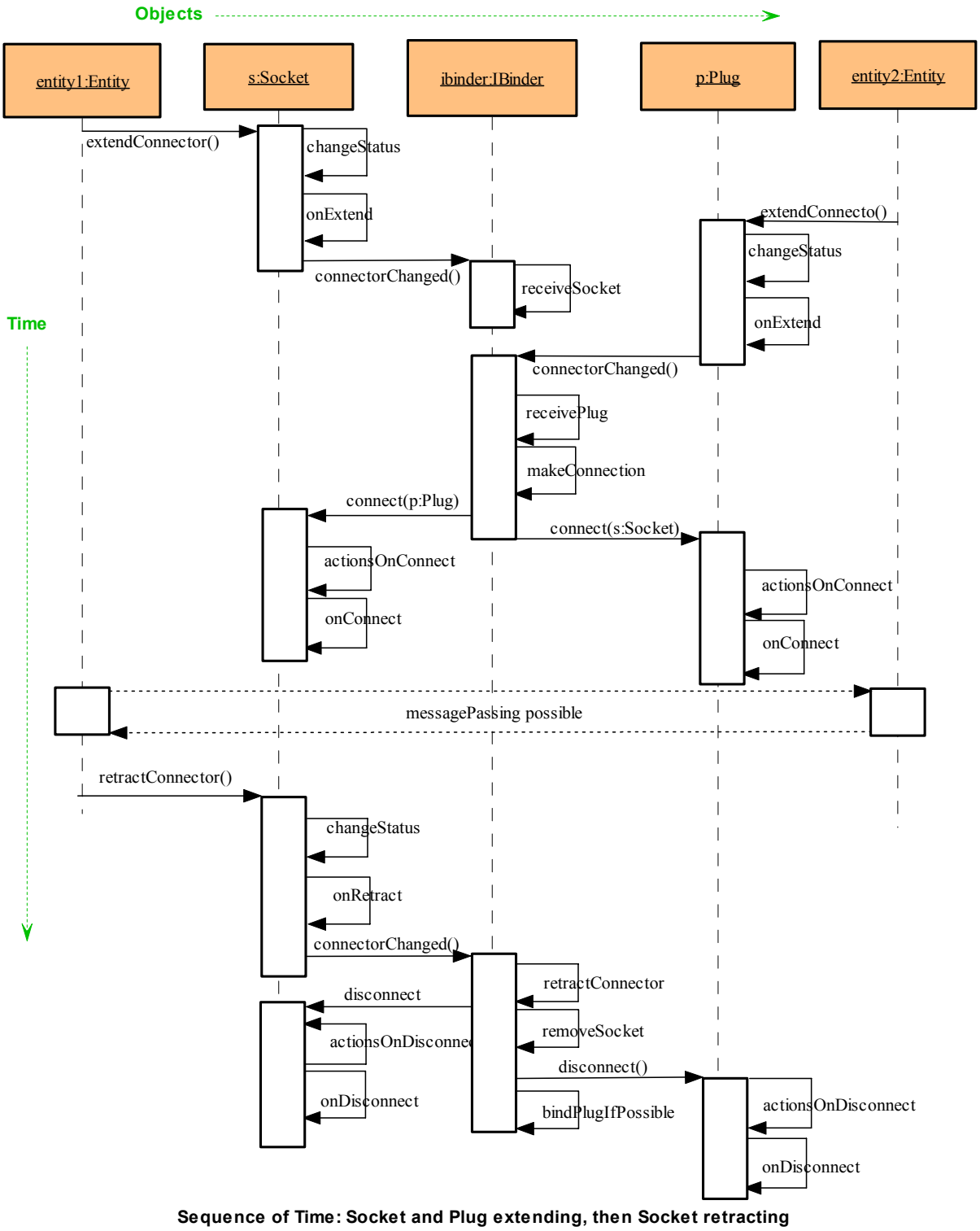


Figure 22. A sequence diagram of Socket and Plug connecting and disconnecting over time.

F. SUMMARY

This chapter introduced the concept of iconnectors. Throughout the chapter the actors and infrastructures, as depicted in Chapter III, were presented in the iconnector notation. The concept of iconnectors permits the model depicted in Chapter III to implemented as a software simulation system.

The infrastructure components were presented in this chapter in a connector notation. Actors were presented at the interface level, but their reasoning is still a black box.

The subsequent chapter describes an implementation of actors for a multi-agent implementation of the computational model of IA. This implementation presents a connector-based agent architecture that permits the software actors to reason and interact with elements in the society.

Next, an evaluation of this model is conducted by mapping the connector-based model to an empirical IA model. This is followed by a detailed discussion of mapping actual and theoretical IA incidents to this model, and a description of their implementation in a multi-agent simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

V. THE STIAM CONNECTOR-BASED AGENT ARCHITECTURE

A. INTRODUCTION

This chapter introduces agent technology. Agents are used in this dissertation to represent actors in the IA model. Next, the connector-based agent architecture is introduced. This is followed by an introduction to the agent's internal mechanisms, followed by the agent's role set and goal structure. Finally, the agent's capability set and behavior moderators are introduced.

B. OVERVIEW

Multi-agent simulations consist of numerous high-level autonomous software entities, called agents, operating in a common, shared environment. The agents in this "outer environment" interact with one another and the objects in the environment. They sense their environment, interpret the sensory input and make decisions as to what actions to perform. These actions in turn affect the environment either directly through agent-to-environment interactions or indirectly through agent-to-agent interaction. Figure 23 depicts these situated autonomous agents that interact with other agents and objects in an external environment [Hiles *et al.*, 2001], [VanPutte *et al.*, 2002].

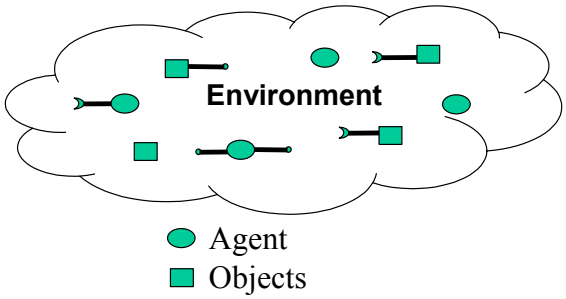


Figure 23. Agents and Objects operating in an external environment.

When building macro-level simulations with these agents, researchers are not interested in modeling cognitive behavior per se; they are interested in how the environment operates as a whole, i.e. as the sum of many parts. Complex, cognitive agents that simulate human reasoning are not appropriate for this level of abstraction [Axelrod, 1997]. Instead, "cognitively limited" [Prietula, 1998] software agents whose

scope is restricted and whose outward behavior *appears* intelligent are employed. This abstraction allows researchers and analysts to gain insight into the evolutionary pattern of the entire simulation environment, while making a complex domain, such as IA, tractable and manageable.

This chapter introduces the agent architecture developed for STIAM. These agents combine the Composite Agent architecture proposed by John Hiles [Hiles *et al.*, 2001] with *connectors*. In Chapter IV *iconnectors* are shown to be an *inter-agent* communications mechanism, in contrast to *connectors* which are an *intra-agent* communications mechanism that allows software agents to bring appropriate actions to bear at the right time and in the proper context. The connector-based agent architecture facilitates the building of relatively simple agents with the following characteristics:

1. They can perform actions that appear intelligent.
2. They can interact with objects.
3. They can interact with each other.

Thus, connector-based agents are used to simulate the *actors* in the *society*, as discussed in Chapter III and IV.

C. CONNECTOR-BASED AGENT ARCHITECTURE

There are six defining characteristics of the connector-based agent architecture:

1. Agents are *reactive*, in that they respond to inputs from the environment without any deep reasoning.
2. Agent goals, actions, and tokens are a function of the *roles* the agent is assigned.
3. Agent decision-making is performed through a dynamic goal management apparatus that allows each agent to prioritize its goals and perform actions to pursue its highest priority goal or goals based on its perception of the environment.
4. Procedural problem solving and action selection are handled by a request-response mediation structure called *tickets*. This structure not only permits the utilization of doctrinally correct procedures, but also allows

dynamic binding of actions based on context using strongly and semantically typed connectors.

5. All sensors and effectors for the agent are performed through iconnectors and resource messages as discussed in Chapter IV. All actions and messages within the agent are performed through a data structure called *connectors*.
6. A set of behavior moderators has been added to create outwardly observable differences in agent behavior among otherwise homogeneous agents.

The connector-based agent a_i, developed to obtain the above six characteristics, has the following *components*:

$$a_i = < e_i, R_i, G_i, T_i, K_i, BM_i >$$

where:

e_i – *A dynamic internal environment of connectors*

R_i – *Role set*

G_i – *Goal set with appropriate tickets and actions*

T_i – *Token set*

K_i – *Knowledge set of dynamic tickets*

BM_i – *Behavior Moderators*

Equation 13. The Connector-Based Agent Specification.

Refer to Figure 24 for a depiction of the components of the connector-based agent architecture.

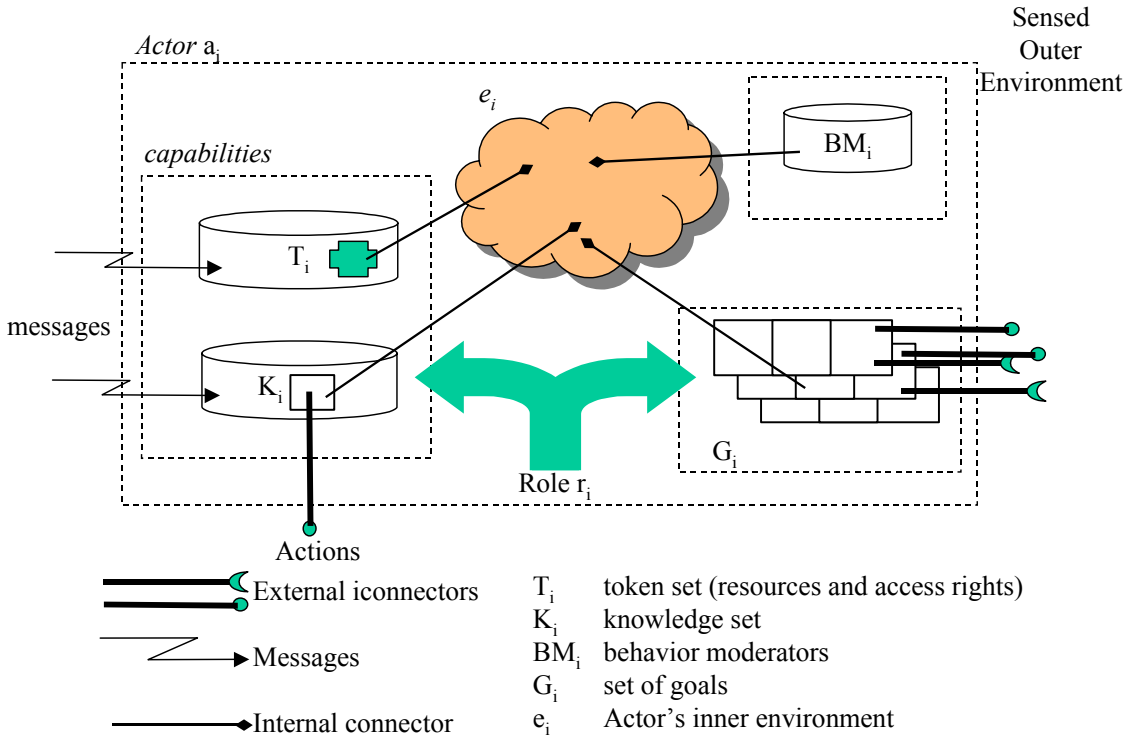


Figure 24. The components of a connector-based agent and their interactions.

D. CONNECTORS AND THE INNER ENVIRONMENT

Just as entities in a society share a common outer environment, components within an agent share a common inner environment, e_i. Similarly, as entities in a society communicate using iconnectors, entities within the agent communicate with a similar, yet simpler mechanism, *connectors*. In implementation, a *Binder* object is created that performs a similar purpose as the *iBinder* in Chapter IV.

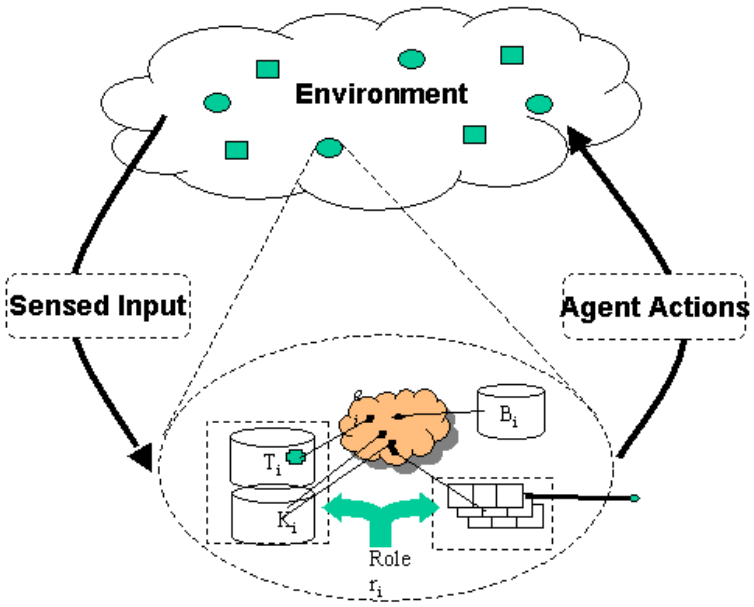


Figure 25. Connector-based agent in an external environment.

Connectors are active objects that sense and react to an environment, just like sockets and plugs in the outer environment. As the agent's inner environment changes, the connectors sense the changes and activate by extending (or deactivate by retracting) in e_i. By attaching connectors to various elements *within* the agent, the connectors signal the element's state of readiness and level of fitness in the current context to other interested internal elements.

Connectors are significantly simpler than iconnectors. Connectors don't have a concept of producers or consumers, nor sockets and plugs. A connector extends into e_i and broadcasts a value. Another connector may extend into e_i listening for a value. If the broadcaster and listener match, then they bind. The connector is defined by:

$$ic_i = < l, s \mid l \in L, s \in S>$$
where:
$$L=\{l, v \mid l = \text{connector label}, v = \text{connector value}\}$$
$$S = \{\text{extended, retracted}\}$$

Equation 14. Connectors consist of a Label and State.

The role of e_i is not unlike *iBinder*; meaning that it acts like an internal switchboard that transcends and binds all agent internal components through these

connectors. Components within an agent create connectors, which they extend into e_i. Each connector has a label that declares its symbolic type. This extension into e_i of a connector with a label therefore signifies an interest in this label. If two components extend connectors with matching labels, then the connectors are of the same type, and the connectors are said to *bind*. When connectors bind, the agent components that created the connectors are notified of the binding and the other component that is party to the binding. The components are also notified whenever the connectors' state or value changes. This simple notification mechanism is a powerful tool for binding the internal components of the agent.

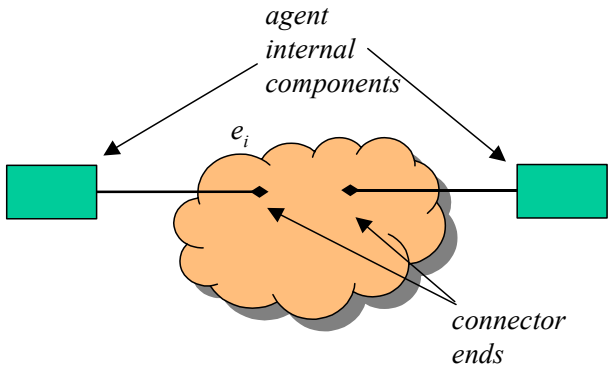


Figure 26. Internal components with connectors extended into e_i.

Connectors bind all of the internal components of the agent, as discussed throughout this chapter.

E. ROLE SET – R_i

The set R_i is the set of all Organizational Roles to which an agent a_i is committed at a moment in time. Each role is defined by a set of one or more goals and capabilities that are specific to the role's behavior or function. As depicted in Figure 24, when an agent commits to a role, it receives a set of goals and capabilities that are then added to the actor's current sets.

F. GOALS - G_i

A goal represents an activity an agent performs to further a role. The set G_i is the set of all goals from all roles assigned to Actor a_i.

At any given time there are numerous goals competing for the agent's attention. Just as humans have multiple, sometimes conflicting goals, an agent too has multiple goals it wishes to satisfy. In human decision-making goals are constantly shifting in priority based on the person's context and state. Klein (1989) showed that experts spend little time generating and analyzing possible courses of action. Rather, they focus on situational awareness, and once a situation is recognized, they execute actions in a reactionary manner. Agents can mimic the flexibility and substitution skills of human decision-making using a variable goal management apparatus within the agent. It is from this goal apparatus where contextually appropriate, intelligent behavior emerges.

1. Goal Structure

A goal has five components:

$$g_i = < s, mm, w, tp, AC>$$
where:
$$s = state \in \{inactive, critical, ready, active\}$$
$$mm = measurement\ method$$
$$w = weight$$
$$tp = threshold\ pair$$
$$AC = action\ set\ for\ achieving\ the\ goal$$

Equation 15. A goal definition.

Each of these components is discussed below.

a. Goal state

The goal state indicates the current condition of the agent's goal. The goal state may be one of these four enumerated values:

- **Inactive** – the goal is currently not considered important to work toward.
- **Critical** – the goal is considered important, but there are no actions that can be performed to pursue the goal.
- **Ready** – The goal is critical, and there are actions the agent can perform to pursue this goal, but the goal has not been selected for execution.
- **Active** – the goal is critical and an action is currently being performed to pursue the goal.

The use of goals and goal state transitions are discussed in detail in Chapter V. Section F.1.e. entitled "Goal Action Set. "

b. Measurement Method

Agents need a way to determine if a goal is critical, i.e., should the agent spend resources pursuing the goal. This is accomplished through the measurement method. The measurement method translates the sensory input received by the goal into a quantifiable measure of the "criticality" of a goal at that instant in time. The measurement method typically uses an algebraic formula and returns a *measurement value*. The lower the measurement value, the less the agent is satisfying that goal, and the more important the goal is to the agent at that moment in time. This dynamic measurement of goal satisfaction permits prioritization and adjustment of goal states based on context.

Goals can be conceptually thought of as active entities. These entities may have connectors listening to e_i that observe the internal state of the actor. These entities may also have external sockets, plugs, or listener iconnectors that extend into the outer environment. From these internal and external connectors the goal measurement method returns its perceived criticality, as shown in Figure 27.

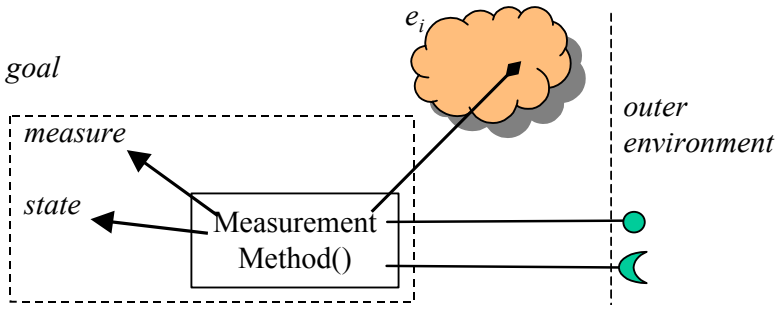


Figure 27. A goal receives input from e_i and the outer environment, and produces a state, measure, and actions that effect the outer environment.

c. Weight

The goal weight is a relatively static, quantifiable value indicating the importance of the goal to the agent over time. For example, a particular *system administrator agent* may believe that a goal of "*protecting organizational information*" is more important than "providing user functionality," and therefore the first goal will have a higher weight than the second.

Goal weights may be modified by having special goal actions adjust the goal weights. This may simulate an agent attending training, or experience in a field. For example, a *system administrator agent* may perform an action of *attend security training* that reinforces the importance of security in day-to-day activities, so the agent's goal weight of *protecting organizational information* may be increased.

To contrast, a measurement value is highly dynamic, possibly changing at every simulation cycle. The measurement value is a measure of the actor belief that this goal currently needs attention.

d. Trigger Threshold and Reset Threshold

The threshold pair consists of a *trigger threshold* and a *reset threshold*. Figure 28 depicts an example where a goal measurement value drops below the trigger threshold at (a), changing the goal state to critical. The goal remains critical until it passes above the reset threshold at (b), where it becomes inactive once more.As demonstrated in Figure 28, when a goal's measurement value drops below the trigger threshold, the goal becomes *critical*. The goal remains critical until the value rises above the reset threshold. This threshold pair provides a simple means for an actor to commit to a goal [Wooldridge, 2000]. Commitment provides a simple implementation of *intent*, or attention holding, in completing a goal.

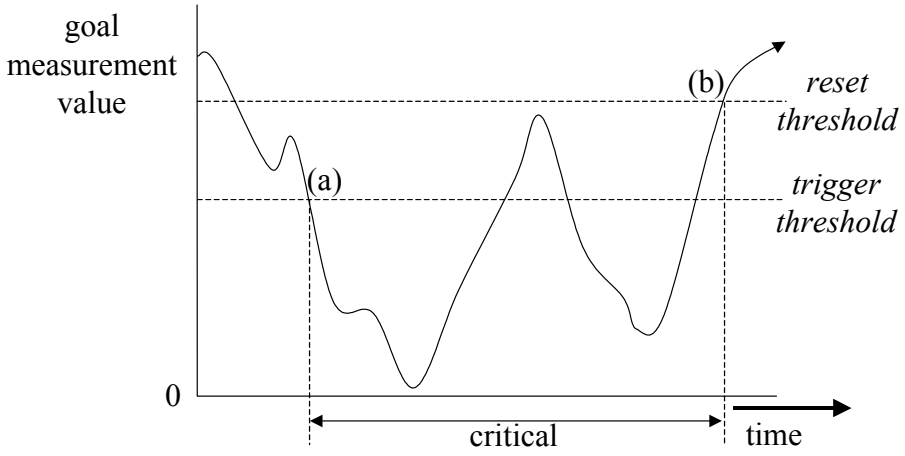


Figure 28. STIAM goal trigger and reset thresholds.

e. Goal Action Set

Tied to each goal is a set of actions that an agent can follow for achieving the goals. Actions are prioritized based on the current perception of the environment. When an appropriate goal is selected for pursuit, appropriate actions are selected for execution. The actual method used for action selection can vary and is discussed later.

When a goal becomes *critical,* it indicates that the actor needs to pay attention to this goal. The goal apparatus examines the goal's action set. If an action exists that can be fired immediately, then the goal state becomes *ready*. If the goal is actually selected for execution, the goal state becomes *active* and actions are executed attempting to fulfill the goal.

The allowed goal states and their state transitions are depicted in Figure 29 state transition diagram.

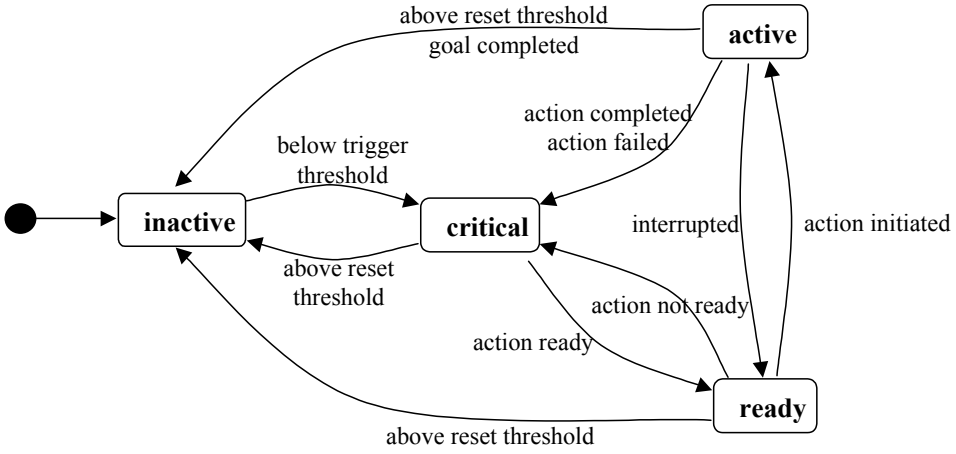


Figure 29. Actor goal state transitions.

Goal switching based on a dynamically changing environment can produce innovative and adaptive behavior [Hiles *et al.*, 2001]; however, it is desirable to constrain goal switching with doctrinally correct and appropriate actions. This constraint is achieved through the encoding of procedural knowledge in a data structure called *tickets.* Tickets are procedural problem solving steps that are encoded into goals, and are discussed later.

2. Goal Manager

Goals are managed through an Actor Goal Manager (AGM). During each simulation cycle the AGM polls the goals and receives the set of goals that are in the *ready* state. For each *ready* goal received, the AGM examines its weight and then executes the next action of the ready goal having the highest weight.

A goal may have more than one action that must be performed sequentially to satisfy the goal and have it return to an inactive state. In this case, the goal becomes the active goal, and continues to execute actions during the agent's execution cycles until it has no further action to perform, is no longer critical, or is interrupted. An example implementation of an AGM with its corresponding goal selection procedure activity diagram is presented in Chapter VII, Section B.3.

The AGM may cause an interruption in the currently active goal and that goal's active action. A goal may preempt the currently active goal and action if the AGM determines that this new goal has a higher weight than the currently active goal.

The action set can be thought of as a set of possible solutions for achieving the goal. Thus, when an agent has committed to a goal, it must then select the best means for pursuing that goal.

Figure 30 depicts a snapshot of an AGM's state at a moment in time. Each row in the table represents a goal and the last entry in the row, labeled *Action Tickets*, contains pointers to action objects. The actions tied to each goal have a Boolean *ready* label indicating whether the action's prerequisites have been met. Goal G4 is the active goal. Prior to becoming the active goal, it had the highest weight of any goal that was in the *critical* state, and had a *ready* action in its action set. Goal G1 is *ready*, in that it is critical and has a *ready* action it can perform if given the opportunity. Goal G5 is critical, but does not have an action it can perform at this time. Goals G2 is inactive: it has an action it can perform, but G2 is not a critical goal. Goal G3 is above the trigger and reset values and is inactive.

Goal Name	State	Measure	Weight	Trigger Value	Reset Value	Action Tickets
G4	Active	.5	.6	.60	.65	T / F
G1	Ready	.2	.4	.29	.46	T
G5	Critical	.4	.2	.45	.45	F
G2	Inactive	.8	.6	.45	.28	T / T
G3	Inactive	.6	.4	.26	.51	F

T = true – prerequisites for action met
F = false – prerequisites for action not met

Figure 30. A snap-shot of a typical actor's goals.

3. Action Set – Tickets and Frames

In order to provide agents with a rich base of procedural knowledge while flexibly supporting adaptive behavior, a data structure called *tickets* was developed [Hiles *et al.*, 2001]. Tickets allow agents to apply procedural knowledge in context. Tickets define the agent's action set, i.e., its means to achieve its goals. They are used to organize procedural knowledge and provide the ability to balance doctrinal behavior with adaptive, innovative action, resulting in enriched problem-solving behavior.

The actions tied to the agent's goals are actually tickets that define how to achieve the goal. Tickets are composed of one or more frames. A frame can be thought of as a container that holds a problem-solving step for a ticket. The frame may hold another ticket, an action to perform, or can be a slot that can dynamically link to an action or ticket at runtime. Tickets are depicted graphically as a sequence of squares arranged horizontally, where each square represents a frame within the ticket. As shown in Figure 31, actions are depicted as independent squares, with connectors and iconnectors possibly extending from the action.

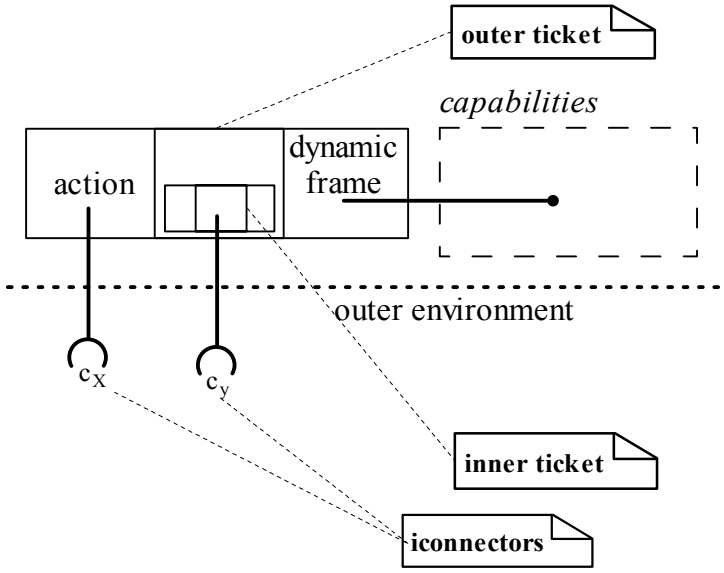


Figure 31. An example ticket.

Tickets may have prerequisites or co-requisites that must be met in order for a ticket to be active. The prerequisites are specific to each ticket, and may include connector values, possession of tokens, or external iconnectors that must be bound. When the AGM is determining a goal's state, it queries the goal's tickets to see if they are able to execute. The ticket prerequisites are checked to determine if it is able to execute at that instant. This may require a recursive call since tickets may need to check member frames that may themselves be composed of tickets, as depicted in the center "inner ticket" frame of Figure 31. The prerequisite function checking returns true if the prerequisites of the ticket have been met.

Each ticket has a measurement method. This method returns a zero value if the ticket determines that its prerequisites have not been met. If the prerequisites have been met, the ticket cycles through its frames and selects the next frame to execute, returning the measurement value of that frame.

Tickets have a weight reflecting their perceived usefulness in solving the goal. The agent examines the set of tickets that have their prerequisites met, and selects the one with the highest weight.

Tickets can have two types of frames: static and dynamic.

a. Static Frame

A static frame is a hard-coded, predefined problem-solving step within a ticket. These frames can be thought of as a phase in a doctrine, tactic, or procedure to solve a problem. A static frame is functionally equivalent to a production, or if-then action rule, that may execute the conclusion (action) if the premise (prerequisite) evaluates to true.

A static frame may hold a ticket or an action. A ticket within a frame can represent a sub ticket, or a sub problem step within a larger problem. An action in a frame is a behavior, activity, or tool execution that is performed by the actor. These actions may cause effects to the agent, or another entity in the outer environment. All actions are connectors or iconnectors that are extended or retracted from the action.

b. Dynamic Frame

Simply encoding procedural knowledge and linking it to various goals is not sufficient for creating robust behavior. The desire is to apply the most appropriate procedures for a given situation. In a dynamic, concurrent system, the "given situation" not only changes constantly, but also is complex, so the system designer can't necessarily conceive of (or account for) every possible combination. Therefore, the mechanism for determining the "most appropriate" procedures must be flexible and able to support the same level of complexity as the changing contexts of the dynamic system. The ability to reference an action commensurate with the situation is provided by allowing connectors as components of dynamic frames.

Connectors are created by dynamic frames and extended into the inner environment, e_i. The connector's value represents requirements that must be met in order to bind a problem-solving step to a frame in a ticket. This permits the ticket to dynamically bind an appropriate action at runtime based on the simulation context. The tickets and actions that bind to these dynamic frames are stored in the agent's *Knowledge Set*, and are discussed later.

As an example, Figure 32 depicts a frame within a ticket tk_1 that requires two connectors labeled c1 and c2. *Action i* does not contain both prerequisite connectors,

so it can not bind. *Action j* contains both prerequisite connectors so it may bind. If ticket tk_1 was selected for execution then *action j* is executed, resulting in iconnector *c4* being extended into the outer environment.

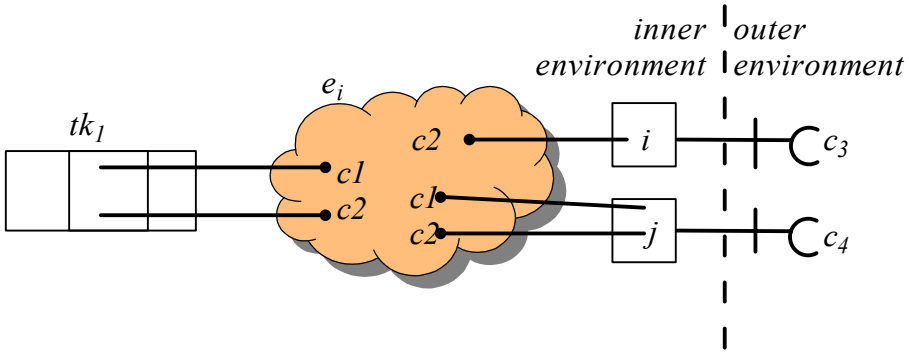


Figure 32. A Ticket tk_i can dynamically bind to Actions j, but is not able to bind to action i.

With the connectors continually reacting to the environment, behavioral and procedural knowledge (tickets) can bind at runtime to fit the context as it develops. This binding is based not only on the state of the environment, but also on the goals of the agent, its capabilities, and its social interactions with other agents. In this way, the correct procedural knowledge can be brought to bear in the appropriate situation.

G. AGENT TOKENS AND KNOWLEDGE SET-- CAPABILITIES

In addition to tickets attached to goals, the actor has several *toolboxes* from which additional resources may be retrieved. These toolboxes consist of the token set and the knowledge set. The elements in these *toolboxes* extend connectors into e_i advertising their existence to the actor. Dynamic frames are then able to bind to these capabilities in order to achieve their goals.

1. T_i – Token Set

The Actor's *tokens* T_i represent the collection of all initial tokens, tokens received from roles, and tokens received from messages. If a frame requires a token it extends a connector into the e_i that will return an appropriate token if one exist in T_i. Additionally, goals have access to the entire set of tokens for attaching to plug connectors being extended into the outer environment (as described in Chapter IV).

2. K_i – Knowledge Set

The knowledge set K_i represents procedural problem solving capabilities and declarative knowledge that are able to bind to dynamic frames of agent tickets. This knowledge set provides the actor with tickets and actions to be used to achieve a goal. Unlike the action sets defined with a ticket, the knowledge set elements bind to frames dynamically at runtime. This is accomplished when a frame extends connectors into e_i, and the inner environment e_i returns an action from the knowledge set that matches the connectors.

3. Agent Learning

In some instances, we may wish to *simulate an actor* learning new skills or acquiring new capabilities. Actor learning can be simulated through the dynamic addition of tickets and tokens during execution. Adding tickets and tokens has the effect of increasing the agent's capabilities and knowledge. Learning is simulated by sending a message to an agent with the additional capabilities within the message. The actor parses the message and adds the capabilities to the agent's token set or knowledge set.

In other cases, we may want an *agent to autonomously* improve its performance over time. In this case, agents can discard tickets that do not further their goals, and increase the use of tickets that have proved successful in reaching goals. This can be accomplished by observing ticket behavior and adjusting ticket weights when the agent observes that tickets succeed or fail. This behavior serves as a simple reactive learning system where the agent learns from the environment, based on "what works" with no human expertise or intervention [Holland, 1996].

H. BEHAVIOR MODERATORS[4]

The behavior moderators (BMs) are subtle differences in individual agents, represented as values that can cause changes in an actor's behavior. The rationale for including behavior moderators is to capture a variety of attributes needed for describing human actors in an IA environment. The BMs are used by the goal apparatus to

[4] The term *behavior moderator* is borrowed from [NRC, 1998].

"personalize" the agent's goal prioritization, thus creating outwardly observable differences in actor behavior. Researchers can adjust the agent's personality, skill, and emotion values and observe how this changes the agent's behavior.

Three categories of BMs are included in a STIAM agent: observable personality, skills, and emotional state.

1. Observable Personality

The agent's Observable Personality values represent a relatively static set that defines that actor's long-term behavior. These values include propensities for risk, loyalty to organizations, ethics, and ambition.

2. Skills

Skills represent a highly abstract set of ability values that an actor possesses. These skills include organizational, social, information technology, security, and management skills.

3. Emotional State

The actor's emotional state consists of a set of attributes that represent the actor's current internal state or *feelings* at any instant in time. These attributes include the agent's feeling of loneliness, security, self-worth, excitement, and fatigue.

Behavior moderators are initialized when an actor is created. These moderators are implemented as connectors, and are extended into the actor's inner environment. An actor may have actions that modify the BMs based on sensor input, thereby simulating education, changes to emotions, etc. These modified connectors alert components that are listening to the connector, causing effects internal to the agent.

The primary use of the behavior moderators are as coefficients to specific goal and ticket weights. BMs may bind to goals and tickets thereby modifying the actor's goal and action selection, creating observable differences in otherwise homogeneous actors.

The moderators selected do not represent a scientific coverage of possible moderators. This is left for future work.

I. LIMITATIONS OF STIAM AGENTS

The STIAM agent architecture as presented has several limitations: the effects of linear problem solving and an inability to learn from success and failures.

The early history of artificial intelligence produced numerous systems that suffered from linear problem solving, such as GPS [Newell and Simon, 1963] and STRIPS [Filkes and Nilsson, 1971]. These systems divided problems into sub problems and solved each sub problem. Difficulties occur when a later sub problem *undoes* the effect of a previously solved sub problem. ABSTRIPS overcame the difficulties of linear problem solving using a *procedural net*, where "a plan is a partial ordering of actions with respect to time" [Sacerdoti, 1974]. Since actions were not ordered sequentially, they could be executed in a way that overcame the linear problem. Likewise, Sussman defined the "Prerequisite_Clobbers_Brother_Goal", where a prerequisite of one goal causes the failure of another [Sussman, 1974]. Tickets and reactive actions have the potential to cause gridlock within a single agent or between two agents, whose actions cancel the effects of previous actions. This difficulty was not addressed in the current connector-based model.

Additionally, there is no learning or historic element in the STIAM agents. This ability would permit the agent to learn from success and failure, and improve its capabilities over time. Both of these challenges are left as future work.

J. SUMMARY

The connector-based architecture facilitates the creation of complex agent behavior through relatively simple components. In later chapters, it is demonstrated that this relatively simple, reactive agent architecture can bring rich, complex adaptive behavior to the computational model of IA. The simplicity of the agent allows researchers to focus their attention on the environment being simulated, and not on the implementation mechanism.

Chapter VII provides implementation details of actors created for the computational model. Actual scenarios are implemented and analyzed in Chapter VIII.

VI. MODEL VALIDATION

A. INTRODUCTION

This chapter provides a validation of the STIAM computational model as a *hypothesis generator* for organizational level IA. While a hypothesis generator cannot predict with accuracy what will occur in an environment, they can be used to generate sequences of events that may possibly occur, and these sequences can be used to perform *inductive reasoning* about the environment under investigation.

To validate STIAM, this chapter demonstrates that the STIAM model captures all of the vital characteristics to the field of IA. A mapping of the elements and relationships of a security model based on empirical data of computer security incidents to STIAM is performed.

Next, this chapter discusses the shortcomings of *functional* models and the advantages of the *concurrent* computational STIAM model.

This chapter provides a validation of the computational model, demonstrating that the elements of the information assurance field can be adequately represented in STIAM.

B. INFORMATION ASSURANCE AND HYPOTHESES GENERATORS

1. Models and Simulations

The term '*model'* means different things to different people. To some *model* refers to a physical reproduction of an entity or environment, such as a toy ship or a diorama. To others, model represents an analytical specification of assumptions, definitions, and equations used to discuss a particular phenomenon or theory [Nelson, 1998], such as Newtonian mechanics or computer system security properties [Bell and LaPadula, 1973], [Biba, 1977], and [Graham and Denning, 1972].

For the remainder of this chapter, the term model refers to a *computational model*. A computational model is a specification of the key entities in an environment, along with their behaviors and interactions, which can be represented by a computer program to explore specific aspects of the environment. A '*simulation'* refers to a computer software

implementation of the model to observe an abstraction of the environment as specified in the model.

2. Induction and Hypothesis Generation

A hypothesis generator is a simulation that reveals possible events and situations that may occur in an environment based on assumptions in the model [Hodges and Dewar, 1992]. The hypothesis generator is a simulation using software representations of entities found in the real world that simulates how these entities interact. A researcher uses inductive logic, reasoning from the specific to the general to generate theories about the world. He examines the output of the simulation, looks for patterns, and generates hypotheses about the real world [Axelrod, 1997]. The researcher might then examine the real world to confirm or deny these hypotheses.

Many of the generated hypotheses may be obvious. When others cause the researcher "to be moved to learn something about the world, the model may then be said to provide insight by poking him (the researcher) to go look at something …" [Hodges and Dewar, 1992].

A hypothesis generator may not actually create a hypothesis in the strict mathematical sense. It may produce a series of events that the researcher can examine, and from this, the researcher can conjecture as to the likelihood of the events. The researcher can then examine the real world and attempt to prove the conjecture, adding these new theories to the researcher's collection of domain knowledge. These theories may then be used by the researcher to further understand the environment [Axelrod, 1997].

It is important to contrast a predictive simulation with a hypothesis generator. The outputs from a validated predictive simulation are potentially observable events or quantities where their predictive accuracy can be measured in a real environment [Hodges and Dewar, 1992]. For example, a validated predictive astronomy simulation may state that a planet X will be at position Y at time T. A hypothesis generator on the other hand "does not give power to see into the actual situation, only into the assertions embodied in the model…it does so not by revealing truth about the world, (but) by

revealing key features of it's own assumptions and thereby causing its user to go learn whether those key assumptions are true." [Hodges and Dewar, 1992].

3. STIAM

As stated earlier, STIAM is a hypothesis generator, not a predictive simulation. STIAM provides a means to create a computational system representing relevant IA characteristics that will help security analysts generate hypotheses and theories about the IA domain.

The next section provides a mapping of a security model based on empirical data of computer security incidents to STIAM. This mapping demonstrates that STIAM contains all of the elements and relationships contained in the empirical model. This validates that STIAM can adequately model all of the elements found in the IA domain.

C. EMPIRICAL MODEL OF INFORMATION ASSURANCE (IA)

Various agencies and organizations collect data on information system security and cyber-crime incidents. The most comprehensive, open-source collection of information has been compiled by the CERT/CC.

1. CERT/CC

The Computer Emergency Response Team/Coordination Center (CERT/CC) is a federally funded research and development center responsible for the study and handling of Internet security vulnerabilities and incidents [CERT, 2002]. Government and commercial entities report information system security incidents to the CERT/CC, and receive assistance in dealing with these incidents.

Howard conducted an analysis of Internet incidents reported to the CERT/CC from 1989 to 1995. He categorized the incidents, and from this developed a *Taxonomy of Computer and Network Attacks* [Howard, 1997]. This taxonomy provided coverage of all incidents in the CERT/CC database.

The model he developed from this analysis is summarized in tabular form in Figure 33. This model demonstrates how an *attacker* uses *tools* to provide *access* creating unauthorized *results* that furthers the attacker's *objectives*.

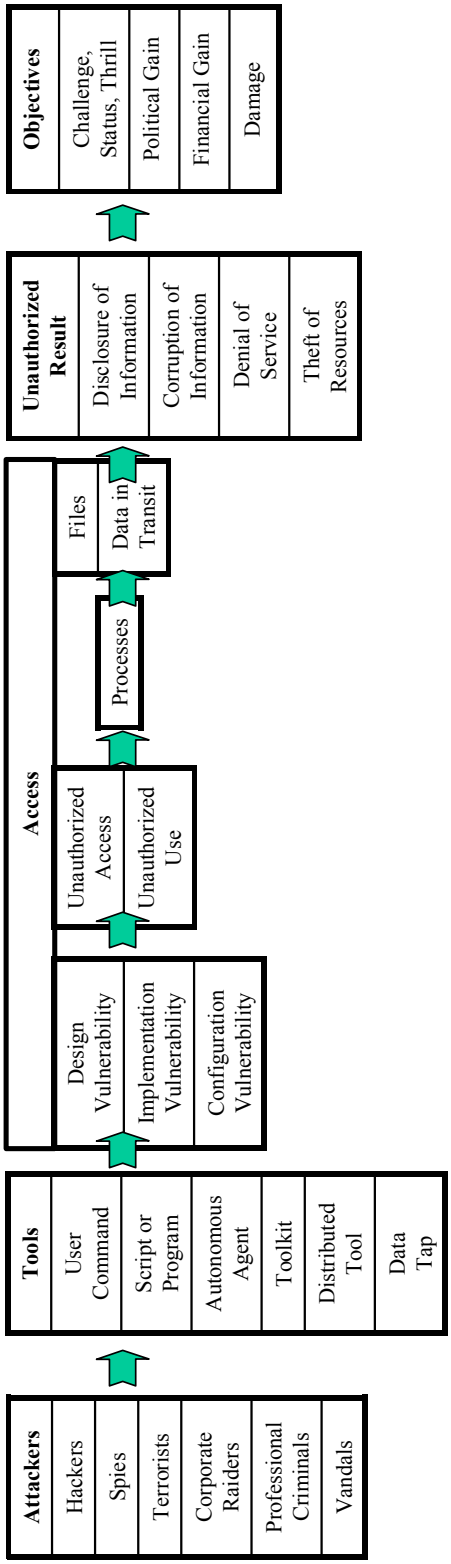


Figure 33. Howard's *Computer and Network Attack Taxonomy*, [Howard, 1997].

Howard's research was based on the Internet incident data of the CERT/CC. As such, his taxonomy provides complete coverage of the CERT/CC database, but the database did not provide complete coverage of security incidents. Any incidents that were not shared with the CERT/CC, did not involve the Internet, or that were not technical in nature were not included in the CERT/CC database.

It has been observed that a large number of cyber attacks against government agencies and corporate organizations are not reported [Minehart, 1998]. For many organizations, reporting successful cyber attacks can damage the perception of the organization in the eyes of customers and clients. In commercial enterprises, the "personal relationship with the customer is the most cherished asset", [Minehart, 1998] and so reporting successful attacks against a corporate information infrastructure may damage the reputation of the corporation, and under most circumstances harm the corporate bottom line. Additionally, reporting security incidents can provide useful feedback to an attacker, possibly releasing additional technical details to the attacker [Minehart, 1998], and even inviting "copycat" attacks.

The CERT/CC data also fails to account for the professional attacker. Professionals are differentiated from amateurs by the effort, cost, and sophistication of an attack[5]. A professional can expend the funds and afford to wait for the right moment to attack, using sophisticated and nearly undetectable methods, such as the subversion[6] of software. For examples of this professional threat see [Myers, 1980], [Karger and Schell, 1974], and [Anderson, 2002]. These attacks are nearly impossible to discover, and therefore are unlikely to be reported to the CERT/CC.

In an effort to develop a more complete taxonomy of security incidents, Howard and Longstaff developed a model based not only on empirical data, but also on general observations and experience in the field of IA.

---

[5] From a personal conversation with William Murray at the Naval Postgraduate School, Monterey, California, December 2001.

[6] Subversion refers to the "covert and methodical undermining of internal and external controls over a system lifetime to allow unauthorized and undetected access to system resources and/or information." [Myers, 1980].

2. Enhanced Model

Howard and Longstaff updated Howard's original work to develop a "Common Language for Computer Security Incidents" [Howard and Longstaff, 1998]. This work developed a "taxonomy of high-level terms and relationships to classify security incidents" [Howard and Longstaff, 1998]. This work added elements to Howard's original work and provided additional coverage of security incidents based on their experience in the security community. They added the ability to model physical and social attacks, rather than just logic-based attacks[7]. They also modified the model, increasing its usefulness and complexity, by adding *action* and *target* categories.

Howard and Longstaff's goal was to "develop a minimum set of 'high level' terms, along with a structure indicating their relationships (a taxonomy), which can be used to classify and understand computer security incident information" [Howard and Longstaff, 1998]. Certainly any model that claims to represent IA must include these elements. By mapping these elements to STIAM, an evaluation of the STIAM model can be conducted.

[7] Physical attacks refer to the theft, destruction, and/or damage of materials. Social attacks refer to manipulating individuals to achieve a goal. Logical attacks refer to manipulating data in an electronic format.

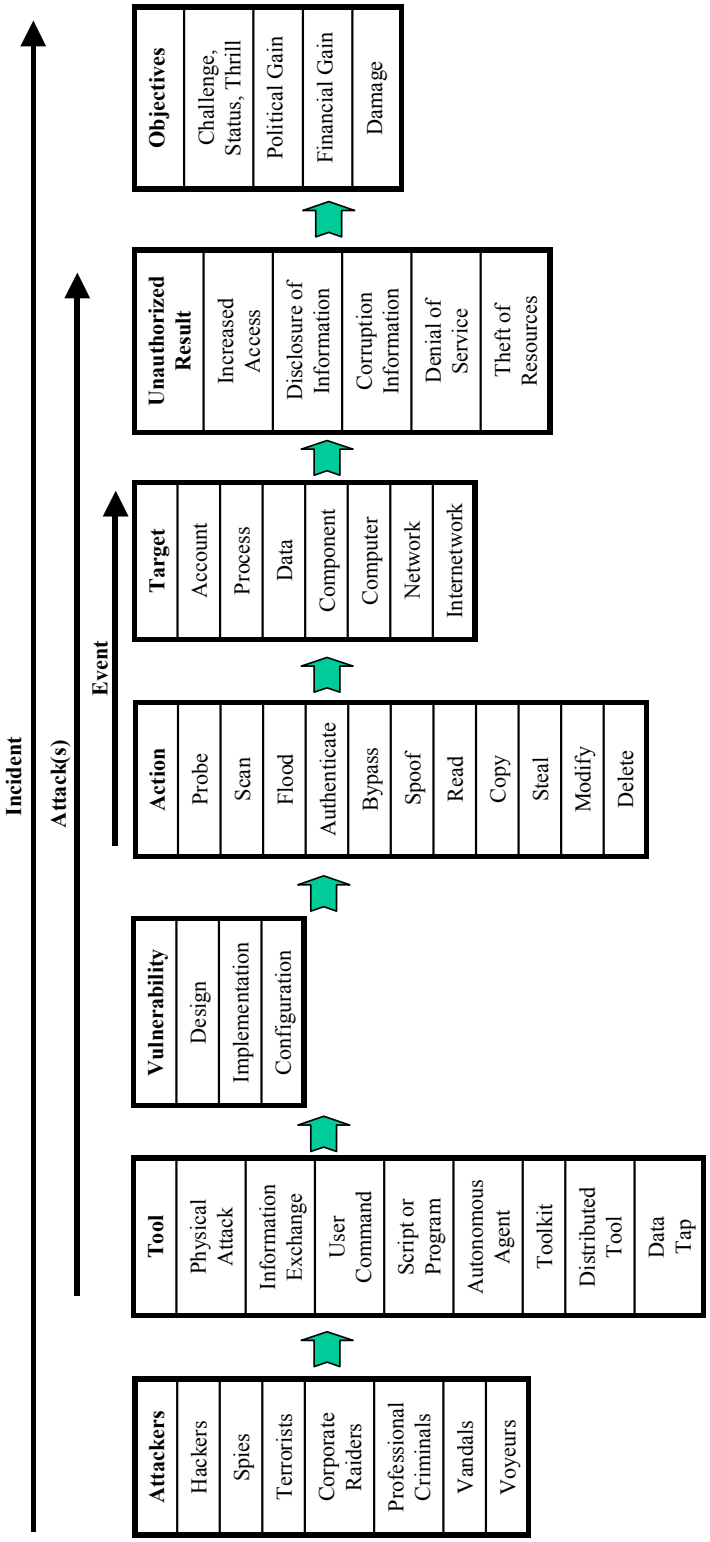


Figure 34. Howard and Longstaff's Computer Security Incidents [Howard and Longstaff, 1998].

D. MAPPING OF EMPIRICAL MODEL TO STIAM

The following is a mapping of the elements of the Howard and Longstaff model, as shown in Figure 34, to STIAM. This will demonstrate that STIAM contains a superset of the elements of this empirical model.

1. Actors and Objectives

Howard and Longstaff's *attackers* and *objectives* represent a subset of the actors and goals in the IA environment. Particularly, they are the individuals who perform malicious actions against computer systems. As stated earlier, Howard and Longstaff's seven 'attackers' provides a simple means/motive label at an instant in time.

STIAM may include any actor goals that a researcher may feel is important, and as such, Howard and Longstaff's attackers and objectives are included in STIAM.

Additionally, actors not listed by Howard and Longstaff may perform actions that affect an organization's information and systems. An example is a benevolent individual who performs some action out of *ignorance* that inadvertently affects organizational information security adversely. Thus, STIAM can model any malevolent actor (attacker), as well as a wide variety of other actors, goals, and actions that can affect the IA of an organization.

2. Tools

Tools represent the means at the actor's disposal to exploit an infrastructure or actor capability. These can be represented in STIAM by tickets and frames in an actor's knowledge pool that can extend iconnectors from the actor. If an actor has a ticket or frame and a desire to utilize this ticket or frame within the context of an active goal, the actor extends the appropriate iconnector into the environment, representing the use of the tool.

All of the tools are defined and discussed in Table 3 entitled "Mapping of Howard and Longstaff's tools to STIAM". This table demonstrates how to model each tool discussed in Howard and Longstaff directly to a STIAM ticket or frame.

Means	Definition[8]	Implementation in STIAM
Physical Attack	*A means of physically stealing or damaging a computer, network, its components, or its supporting systems (such as air conditioning, electrical power, etc)*	An infrastructure that is susceptible to a physical attack has a socket with tokens indicating appropriate physical proximity. An actor who has these tokens and the desire and capability to perform a physical attack may extend a plug connector, causing the bound socket to effect the infrastructure. A researcher may model a physical attack against another actor using a similar method.
Information Exchange	*A means of obtaining information either from other attackers (such as through electronic bulletin boards), or from the people being attacked (commonly called social engineering)*	These are modeled as a plug connector that binds to an actor or infrastructure. The binding results in a message being sent to the attacker that contains the newly acquired information. This information may be access rights, as tokens, or procedural knowledge, as tickets.
Commands, Script or Program, Toolkit	*Exploiting a vulnerability by entering commands to a process through direct user input …(or) through the execution of a file of commands (script) or a program at the process interface. This also includes software packages, which contain scripts, programs, or agents to exploit vulnerabilities.*	These are modeled as a socket or plug connector between an actor and an infrastructure. This may result in an infrastructure-to-infrastructure message or binding.
Autonomous Agent	*A means of exploiting a vulnerability by using a program, or program fragment, which operates independently from a user (includes viruses and worms).*	An actor spawns a new 'logical' actor whose life span may be limited. This *child* actor executes tickets assigned by the parent *actor*.
Distributed Tool[9]	*A tool distributed to multiple hosts, which can then be coordinated to anonymously perform an attack on the target host simultaneously after some delay.*	Infrastructure(s) have sockets representing distributed tools. A master sends a message to the zombie sockets causing a plug to extend representing the distributed tool. This plug contains a token representing the number of simulated zombies participating in the attack.
Data Tap	*A means of monitoring the …emanations from a computer or network using an external device.*	A socket on the infrastructure represents emanations; a plug from an actor represents the desire to monitor. The binding causes a message to be sent from an infrastructure to an actor representing the interception of the new data, such as tokens.

Table 2. Mapping of Howard and Longstaff's tools to STIAM.

[8] The definitions are taken directly from [Howard and Longstaff, 1998].

[9] A distributed attack typically has a 'master' who centrally controls multiple 'zombies' on compromised hosts. At the direction of the master, the zombies perform a coordinated attack against a designated 'target' host.

In addition to the malicious tools discussed in Howard and Longstaff, STIAM must deal with tools used by benign actors to simulate individuals performing non-malicious actions. Thus, Howard and Longstaff's tools are a subset of all tools available to actors that can be represented in STIAM.

It is important to realize that tools represent the means to an action, not the action itself. As such, the tools represent a way to place an action on a target. An action has utility only if a compatible connector for the action exists on the target.

3. Vulnerability

Infrastructures have an interface composed of iconnectors to which actors, having the appropriate iconnectors, can bind to perform actions. This interface and the subsequent actions that occur upon binding to this interface define the functionality of the infrastructure. Some of these capabilities are deliberate and known. Others capabilities represent 'unspecified functionalities' or vulnerabilities -- "a weakness in a system allowing unauthorized actions" [Howard and Longstaff, 1998]. In the case of STIAM, vulnerabilities represent a subset of the *sources* of socket connectors. Howard and Longstaff define three types of system vulnerabilities: design, implementation, and configuration of systems; similar to the design, implementation, and maintenance vulnerabilities described in [Myers, 1980].

A special hybrid vulnerability is a capability in a system caused by an incorrect policy specification. The incorrect policy specification may cause an incorrect implementation or configuration of a system, creating a capability in the system not intended by the management, had they known of the inconsistency *a priori*. While this incorrect specification might be caused by ignorance on the part of the management of the organization, we have chosen to place this vulnerability under design vulnerability.

A **design vulnerability** is "a (capability) inherited in the design or specification of the hardware or software whereby a perfect implementation results in this (capability)." [Howard and Longstaff, 1998]. These vulnerabilities are typically attributed to engineers. Real world examples include poorly written protocols, such as the TCP/IP protocol suite [Bellovin, 1989] and the wired equivalent privacy (WEP) protocol [Walker, 2000].

An **implementation vulnerability** is "a (capability) that results from an error made in hardware or software implementation of a satisfactory design." [Howard and Longstaff, 1998]. These vulnerabilities may be the result of a coding or manufacturing error that accidentally introduces a flaw in the system [Karger and Schell, 1974]. As discussed earlier, professional attackers will try to deliberately subvert hardware or software at some point in a system's lifecycle in order to install *unspecified functionality* to the system [Karger and Schell, 1974], [Brinkley and Schell, 1994], [Myers, 1980].

A **configuration vulnerability** is a "(capability) resulting from a configuration of the system" [Howard and Longstaff, 1998]. These vulnerabilities may arise due to an end user not modifying the default settings such as a default account or password, vulnerable services enabled or installed, or global write permissions on newly created files [Atkins *et al.*, 1996]. Additionally these vulnerabilities may be introduced by system administrators or users who incorrectly install or configure software on a system such as accidentally inactivating protective measures and misconfigured routers or firewalls.

When a researcher or analyst is building a model of a real enterprise infrastructure, they determine what the critical resources are within the enterprise. Next, they examine the interfaces of both the resources and infrastructure, and determine how actors and other infrastructures can bind to this infrastructure, thereby defining the capabilities of the infrastructure. Some of the interfaces and their effects on the model are deliberately planned in the real-world system. Others are not, and these unplanned capabilities correspond to the vulnerabilities as defined by Howard and Longstaff.

Howard and Longstaff's three categories provide a catalyst to the researcher's thought process. These vulnerabilities can be modeled in STIAM to help show the researcher the effect on the organization if these vulnerabilities are realized.

While STIAM provides complete coverage of Howard and Longstaff's vulnerabilities, Howard and Longstaff's model fails to provide coverage for social

engineering[10]. In STIAM, as discussed in Chapter IV, an ignorant actor will extend connectors out of ignorance and manipulative actors can take advantage of this ignorance to achieve their goals. Thus, a fourth vulnerability is *ignorance*.

4. Action

An action represents a "step taken by a user or process in order to achieve a result" [Howard and Longstaff, 1998]. In STIAM, an actor executes a tool from within a frame that extends an iconnector, and may bind to a targeted iconnector. This binding may result in an action performed by the owner of that target iconnector. Thus, tools are executed by the attacker, iconnectors may bind, and the resulting action is executed on another entity in the environment.

Howard and Longstaff define eight actions of interest in IA. Due to the level of abstraction of STIAM, two of these actions, probe and scan, are combined.

a. Probe/Scan

Since the *infrastructure* represents the aggregate of all information processing capabilities and resources, probe and scan are combined to 'determine characteristics of an infrastructure'. A probe or scan of an infrastructure is modeled using a listening connector. An actor extends the listening connector, and the actor is notified when a matching iconnector is discovered. A probe/scan is initiated by an actor and operates on a society.

b. Flood

A flood is an overloading of an infrastructure capability, resulting in a potential *denial of service* (DOS). Floods are modeled by a socket iconnector on an infrastructure, typically called a flood socket. If an actor has the prerequisite tool that produced the appropriate plug connector, he can bind to this flood socket causing the retraction of appropriate iconnectors in the infrastructure, representing resources and processes that are no longer available. It is important to keep in mind that when the actor

[10] Social engineering refers to using nontechnical interpersonal deception to manipulate individuals into providing information in order to bypass security controls. See [Winkler, 1997] or [Parker, 1998].

disconnects from the flood socket and discontinues flooding the infrastructure, appropriate restoration and reinitialization actions must take place on the infrastructure to reestablish capabilities as appropriate. Figure 35 represents an infrastructure where an actor is able to perform a flood by binding to the flood socket. This binding to the flood socket causes the resource socket to retract. In this example, when the actor stops flooding, i.e. disconnects from the flood socket, the resource socket extends and the resource is available.

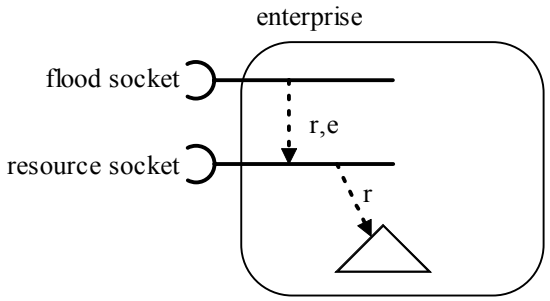


Figure 35. A socket that represents *flooding* a *resource*. Successfully binding to a flood disconnects (retracts) other iconnector.

c. Authenticate

Authentication represents "providing identification to a process (or actor) in order to have an identity verified in order to access a target" [Howard and Longstaff, 1998]. While all bindings can be thought of as an authentication, this action refers to authenticating an actor's identity *in order to receive additional access capability*. This can be modeled by an actor presenting some tokens to an authentication iconnector. If the tokens are accepted, i.e. the actor actually binds to the iconnector, then the iconnector sends a message to the actor, providing an additional token that represents the successful authentication as in Figure 36. Of course, an actor may provide false tokens to the authentication process thereby authenticating a false identity, as in spoofing, discussed below.

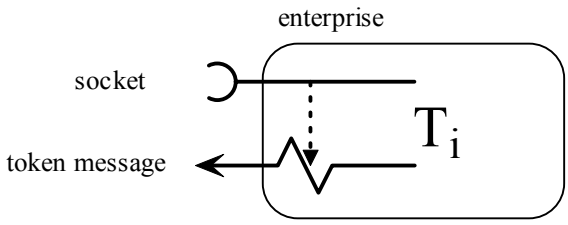


Figure 36. An Iconnector that replicates the *authentication* process.

d. Bypass

A bypass is "avoiding (the) security process by exploiting a vulnerability" [Howard and Longstaff, 1998]. Bypasses can be modeled in STIAM as an interface to a resource or infrastructure that does not require the typically necessary tokens. An actor can bypass normal security if a vulnerability exists on the infrastructure, such as an operating system vulnerability, that can be exploited by a socket.

Figure 37(a) illustrates a normal access method as modeled in STIAM. An actor binds to connector *c1* to replicate the authentication process. The infrastructure provides token T_i that can then be used with connector *c2* to access resource r in the 'normal' (i.e. authorized) manner of access.

Figure 37 (b) illustrates a bypass action in STIAM. An example of an operating system vulnerability is connector c3, that permits any actor *read* or *write* access to resource r, if they posses a token T_j, *knowledge of the vulnerability*.

Figure 37(b) also illustrates a backdoor. Typically, a backdoor must first be activated before it can be exploited. The activation requires an individual to have special knowledge about activating the backdoor. After the backdoor has been activated, an attacker needs special knowledge as to the process of accessing the backdoor to access the critical resources. Connector *c4* represents a backdoor installed in subverted software as an intentional, yet hidden socket that requires token T_l – *knowledge of the backdoor*. The backdoor is initially hidden, and cannot be accessed until an actor binds to *c5*, which requires token T_k, *knowledge of backdoor activation method*, to activate the backdoor. Both of these demonstrate the ability to model a variety of bypass actions in STIAM.

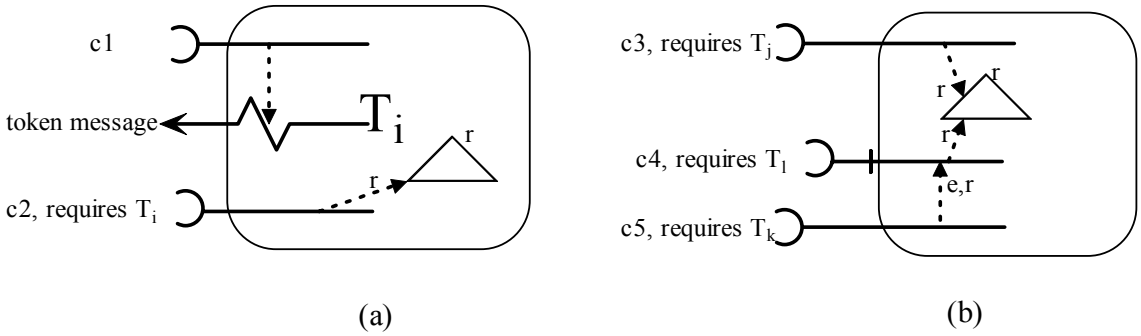


Figure 37. Bypass Actions in STIAM.

e. Spoof[11]

A spoof is nearly identical in implementation on STIAM as an *authentication*. The only difference is that an entity uses tokens on a connector in an unauthorized manner in order to authenticate a false identity, thereby 'tricking' an authentication process.

STIAM can model the four fundamental methods of spoofing:

- **Actor-to-Actor** – An actor presents false tokens to another actor, thereby the attacker tricks the recipients of the spoof into believing the attacker is a different actor. This may represent an example of social engineering.
- **Actor-to-Infrastructure** – An actor presents false tokens to an infrastructure claiming to be another person in order to receive the other person's access rights. This may represent falsifying a system authorization process with acquired passwords or sending 'spoofed' email with a false 'from' address.
- **Infrastructure-to-Actor** – A system claims to be another system to an actor. A real world example is a computer system that does not have a *trusted path*[12] implemented and permits a process to 'pretend' it is a legitimate logon screen in order to capture the user's authentication tokens.

[11] While there are conflicting definitions for spoofing, it is used here to mean "an active security attack in which a machine on the network masquerades as a different machine" [Howard and Longstaff, 1998].

[12] A trusted path is a "mechanism by which a person at a terminal can communicate directly with the (system protection mechanisms). This mechanism can only be activated by the person or the (system protection mechanisms) and cannot be imitated by untrusted software" [NSTISSC, 2000].

- **Infrastructure-to-Infrastructure** – This represents a device spoofing another device. This might represent a *DNS spoof*[13], or configuring a false machine identity, such as an IP address.

f. Read

Reading is "obtain(ing) the content of data in a storage device..." [Howard and Longstaff, 1998]. In STIAM, reading is equivalent to binding to a resource, thereby obtaining the contents of the resource. Reading a token or ticket is performed by binding to a socket and having a token or ticket sent to the actor by a message.

g. Copy

Copying is similar to reading, except that the act reproduces the data and leaves the original unchanged. There are two fundamental types of copying. The first is copying a ticket or token, which is performed by binding to a iconnector and receiving a message that contains the ticket or token.

The second is copying a resource which is also similar to reading in that an actor binds to a iconnector, except here the actor receives a special token through a message. This new token permits the actor to bind to a newly created resource socket that requires this new token.

h. Delete

Deleting a token or ticket results in the object being removed from the entity. Deleting a resource causes all iconnectors to be retracted from the resource. Since no active iconnectors remains, the resource is no longer available. While it may seem that the resource should be removed from the infrastructure, this action is not supported in the basic STIAM model, and in some instances may cause problems. For example, Figure 38 below depicts an infrastructure that supports both delete and backup. When an actor binds to connector c1 a backup is performed on the resource and the actor receives token T_i. When an actor binds to c2 the resource is deleted, perhaps maliciously,

[13] A Domain Name Server (DNS) spoof is performed by sending a network router false network-address data, causing the device to route traffic incorrectly and undetected.

resulting in retraction of the resource iconnector c3. If an actor has the token T_i, then he can bind to c4 and restore the resource from the backup, causing the resource connector to extend.

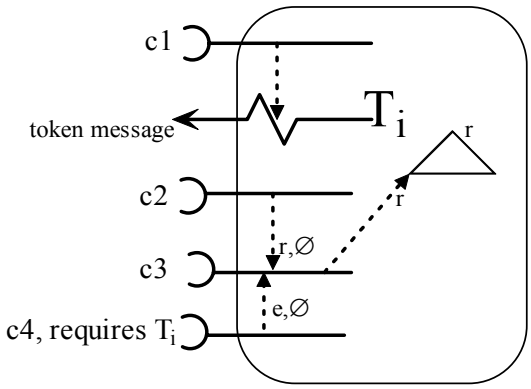


Figure 38. Deletion and backup on STIAM.

i. Steal

Stealing is similar to a copy with an additional action – all other resource sockets that connect to a resource are retracted. In effect, the actor has made a copy of the resource and forbids all others from binding. Examples of a stealing action are copying data to a floppy disk and deleting the data from a system or stealing a notebook computer with a critical resource. Stealing a token can be represented by *copying* the token, then *deleting* it from the original source. Stealing a resource can be represented by copying the resource, then deleting the resource, as in (j) below, but retaining the new extended resource iconnector that was established by the copy.

j. Modify

Modifications occur when an actor writes (binds in a *write* mode) to a resource, in effect changing the version of a resource. Some modifications are authorized and proper; other modifications are malicious. While this is easy to implement in STIAM, there is no easy way to distinguish an unauthorized modification from an authorized modification. The only way to determine this is to examine the actor's goals.

5. Target

Targets are entities to which actions are directed. While Howard and Longstaff defined seven types of targets (account, process, data, component, computer, network, internetwork), they also declare, "The first three are 'logical' entities and the other four are 'physical' entities". This matches nicely to STIAM's concept of resources and infrastructures, respectively.

In STIAM, 'logical' targets (account, process, data) are represented as *resources*, and 'physical' targets (component, computer, network, internet) as *infrastructures*. Howard and Longstaff imply that "logical entities" are computer accounts, computer programs in execution, and electronic data found on computer systems. In STIAM, these are resources, and the STIAM concept of resources in not limited to electronic format. Resources may be in an electronic format, paper, human memory, or other formats. Thus, Howard and Longstaff's logical entities are a subset of all resources in a society that can be represented in STIAM.

Howard and Longstaff indirectly include social attacks. A social attack against an ignorant individual may have an indirect goal of obtaining account information, but the target of the attack itself is on an individual, with the result being increased access. Thus, a third category of target in STIAM is an *actor*.

6. Result

Howard and Longstaff define the "logical end of a successful attack" as the unauthorized result. Example results follow.

a. Increased Access

Increased access results in an actor having the *ability* to bind to additional actors, infrastructures, or resources. This can occur because of additional sockets that have extended due to an action, or an actor receiving additional tokens or tickets from messages that permit additional bindings.

b. Disclosure of Information

This results in an actor actually binding to a resource for which he may be capable but not authorized.

c. Corruption of Information

This results in modifying information – an unauthorized write mode binding to a resource. An organization may have means in place to detect the corruption of information and resources. This may be done with a cryptographic checksum, where a sender computes a checksum value for the information or resource, the receiver computes a checksum, and a comparison of these two values can determine if the information or resource has been modified.

As stated earlier, an actor may desire and perform an act that causes an unauthorized write to a resource, but the effect may not be readily apparent from the graphical diagram, since authorized actors may perform unauthorized write actions that change data in an unauthorized way.

Detecting this result requires the researcher to examine the actor's goals and actions. As stated earlier, if a researcher is modeling accidents in his scenarios, it may be impossible to distinguish some accidents from malicious acts by examining outward STIAM diagrams. In the real world, it may be impossible to tell from observable evidence if an authorized end user incorrectly enters input values or accidentally deletes a corporate database. Likewise, it may not be possible to tell if an actor corrupts an enterprise resource or performs a denial of service without examining the agent's goal structure and actions.

d. Denial of Service (DOS)

This results in the retraction of socket connectors; thereby no other actor is capable of accessing the resource.

e. Theft of Resources

This results in an actor binding to an infrastructure when not authorized. The act could use the resource or tokens as a jumping off point for further attacks.

7. Summary

The sections above demonstrate that all of the elements of a security model based on empirical data, and modified to take into account additional observations in the IA

environment, can be are mapped into the STIAM model. Additionally, numerous additional elements have been identified that are modeled in STIAM, but are not described in the enhanced empirical model. Thus, Table 4 shows that STIAM is a superset of the Howard and Longstaff model, and as such, contains all of the elements of this model.

Key Components	Howard and Longstaff	STIAM
Actors	"Individuals who attack a computer to achieve a (malicious) objective"	Includes all relevant actors in the environment, including benign actors.
Objectives	"The purpose or end goal of an incident" driven by the actor type...static	Includes relevant goals an actor may have based on various assigned roles. These change over time as the actors state and roles change.
Tools	"Means of exploiting a computer or network vulnerability"	These tickets and frames are means to access an actor or infrastructure to achieve their goals.
Vulnerability	"A weakness in a system allowing unauthorized actions"	Includes all sources of capabilities on an actor or infrastructure, including ignorance.
Action	"A step taken by a user or process in order to achieve a result"	An event that occurs as the result of a binding includes malicious acts as well as routine actions that affect the actors, resources, and infrastructures of a society.
Target	"A computer or network logical entity or physical entity"	Includes all of these targets, in addition to social targets (other actors).
Result	"The logical end of a successful attack"	These are the high level interpretations of bindings and messages that result from tools being deployed.

Table 3. Comparison of key components in Howard and Longstaff model and the STIAM model.

E. INFORMATION ASSURANCE (IA) AS A CONCURRENT SYSTEM

Howard and Longstaff's taxonomy accepts a static finite set of inputs and provides a mapping to a static finite set of outputs.

$$F(i) = o$$
$$\text{where :}$$
$$i \in \{attacker \times objective \times tool \times system\ vulnerability\}$$
$$o \in \{action \times target \times result\}$$

Equation 16. Howard and Longstaff's Functional Model.

Systems like these are referred to as functional, or relational, systems [Wooldridge, 2000].

Concurrent systems on the other hand cannot be expressed by a function [Pnueli, 1986]. In a concurrent system, each entity in the system can sense and independently *react* to the environment, which consists of other *reacting* entities [Pnueli, 1986]. Unlike a function that computes some value from a set of inputs and halts, the concurrent system's collection of autonomous entities *react* to each other continuously. Thus, for a given input, it may not be possible to determine an output *a priori*. The entities in a concurrent system must be described in relation to the entity's current state and the state of the surrounding environment.

The IA environment is thus a concurrent system. Actors operate continuously, choosing local actions to perform based on their perception of the environment. Infrastructures and resources are modified by individual actors without other actors' knowledge, providing functionality unknown to the users of the resources.

While at any point each actor has a finite set of actions it can choose from, the actor bases the choice of which action to perform on the observed actions of other actors, and its beliefs about the actions of other actors. The actor can then adjust beliefs and choose another action to perform based on his chosen actions, and the actions and reactions of the other actors [Wooldridge, 2000]. Therefore, *a comprehensive model of IA should not be expressed as a functional model*.

STIAM provides an expressiveness not found in functional models. First, STIAM provides a graphical representation of the instantaneous states of the actors and the environment. This instantaneous state description provides the value of the infrastructure and resource state, in direct comparison with the actor's goals and actions. Second, by viewing the changes in the environment over time, an analyst can get a clearer picture of the dynamic environment. This graphical discovery over time allows the observation of

the evolutionary patterns in the environment. These patterns may provide further insight into where observers need to look in the real environment.

Thus, STIAM can provide a graphical expressiveness with concurrency support that is not possible in sequential functional models. These strengths aid in understanding not only the elements and their interactions, but also where the environment capabilities and vulnerabilities may evolve as a whole.

F. SUMMARY

This chapter explains that it is possible to produce a hypothesis generator and validate this against empirical evidence and experience in the field. The STIAM model accounts for the information assurance elements as identified by empirical evidence. As such, this chapter provides validation of the hypothesis that the STIAM model and simulation can model the IA environment at the organizational level.

Additionally, STIAM is more general purpose than technology-centric models. This generality provides STIAM with the ability to model benign actors, ignorance, and other aspects of organizations, and to examine how these can adversely affect the assurance of an organization's information and information systems.

The descriptive model provides a means of graphically representing the highly complex domain of IA as a concurrent model. This graphical representation provides additional expressiveness not found in traditional functional models, and aids in examining much more complex environments than possible using functional models.

The next chapter presents an implementation of STIAM as a computational system. This system provides researchers with an artificial environment in which to examine the effects of various system and personal vulnerabilities on the information and information systems of an organization. This chapter presents an implementation of several situations that can be encountered in the IA domain. These situations were implemented in STIAM as a proof of principle to demonstrate the utility of the STIAM model.

VII. EXAMPLE SOFTWARE IMPLEMENTATION

A. INTRODUCTION

This chapter presents a proof-of-principle implementation of STIAM. Using this implementation, several small scenarios were developed and encoded to evaluate the model and the software system. Developing the software system provided an abundance of insight into the challenges of IA modeling and multi-agent system (MAS) development.

This chapter begins by introducing the software packages and diagrams, and the system flow for key algorithms. This includes a simulation engine and utilities that were developed to manage the objects and agents in this multi-agent simulation. It also includes the specification of the entities, actor agents, and scenarios needed to run the simulation.

Chapter VIII introduces scenarios that were implemented, an analysis of these scenarios, and a discussion of the significant insight gained from their implementation.

Package and class diagrams are provided in UML. Names of packages and classes are annotated with `fixed-width` font. Class diagrams are presented graphically using Jvision, version 1.2 from Object Insight (http://www.object-insight.com).

B. SOFTWARE IMPLEMENTATION

This implementation of STIAM was developed as a Java application. Java was chosen because of its platform independence, memory management, strict type checking, and object-oriented design, making it a good prototype language.

A total of six packages were developed. The packages are:

- `simsecurity` – contains the simulation engine called `SimManager` that loads simulation scenarios, build the GUI, and executes the main simulation thread. This package also contains the `Token` class that is used across all packages.

- `entity` – contains the `Entity` class, along with the specialized child classes: `Infrastructure` and `Resource.`
- `actor` – contains the `Actor` class and its component classes, such as `Role, Goal, Ticket`.
- `connectors` – contains the `Connector` and `IConnector` classes, along with their respective `Binder, IBinder, Socket` and `Plug`, and `Action` classes.
- `scenarios` – contains XML scenario files and specialized `Actor` and `Infrastructure` classes for use in these scenarios.
- `utilities` – contains basic programming utilities, and data structures necessary for the simulations

Each of these packages are presented below in Figure 39. In this implementation, *organizations* were not explicitly implemented; rather, *roles* were assigned to actors and the actors retained the roles throughout the simulation.

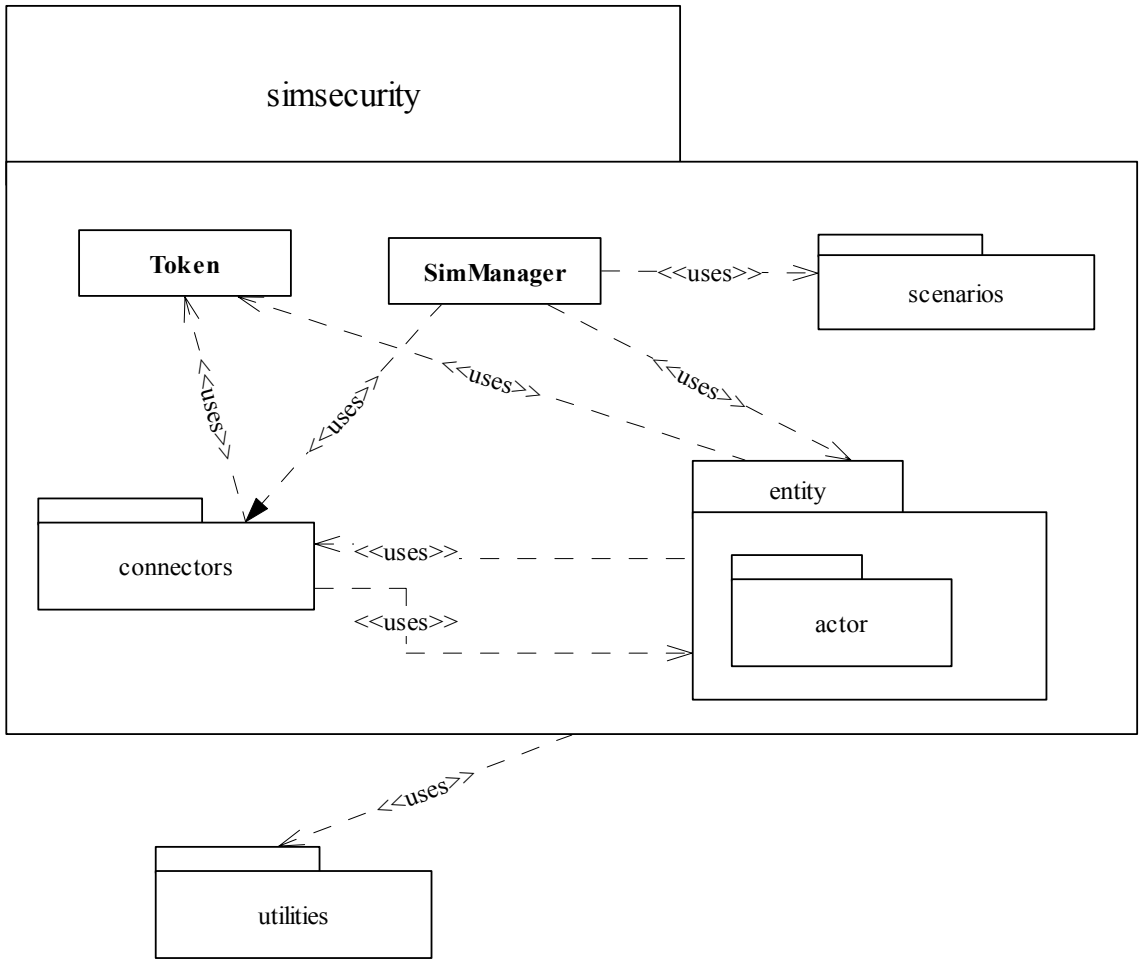


Figure 39. The package diagram for an implementation of STIAM.

1. SimSecurity Package

The `simsecurity` package contains the `scenarios`, `connector`, and `entity` package, as depicted in Figure 39. Since all of the packages use tokens, the `Token` class is a component of the `simsecurity` package.

`SimManager` is a singleton class that manages the simulation. The `SimManager` creates the graphic user interface (GUI), loads a scenario file, instantiates all of the entities declared in the scenario file, and executes the simulation as a single thread. This process is represented in Figure 40.

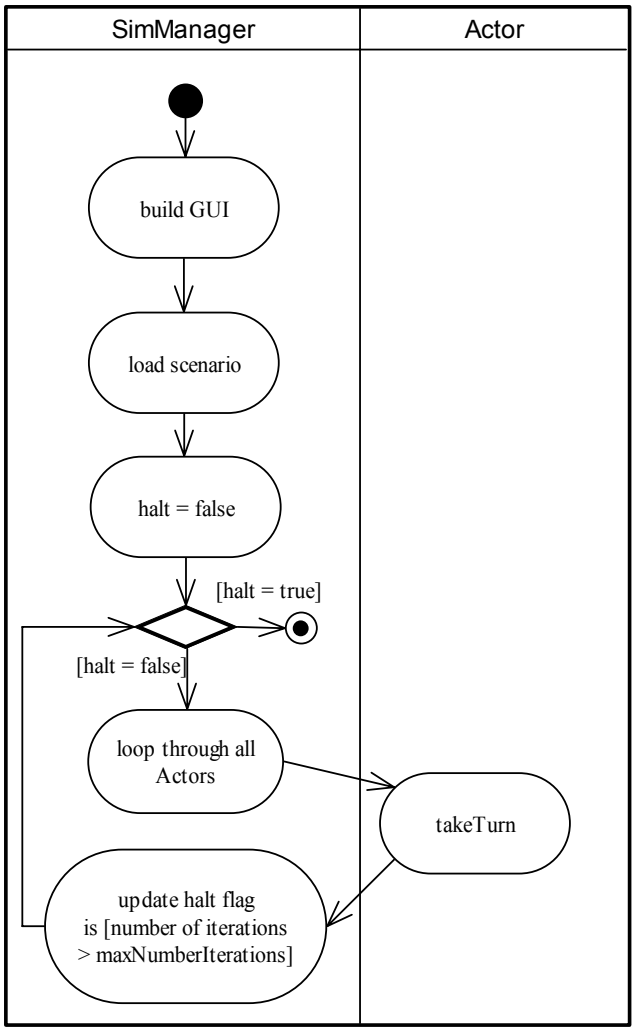


Figure 40.SimManager builds the GUI, loads a scenario, and repetitively loops through all of the Actors.

Scenarios represent defined societies that are represented within the simulation software. Scenarios files are encoded using a customized document type-definition schema for the Extensible Markup Language (XML). Actor and Infrastructure objects are loaded dynamically based on the entity's name matching an existing, compiled class file. See Figure 41 for the activities of the scenario loader.

In this implementation, specific infrastructures are specializations of the `Infrastructure` class and are prewritten and compiled with their component `Token`, `Resource` and `Iconnector` components. If a `Token` is declared in the constructor of

an `Infrastructure` that has not been previously instantiated, an exception is thrown indicating an illegal configuration.

Individual actors are declared as specialization of the `CompositeAgent` class. These actors declare their component `Token`, `IConnector`, `Goal`, and `Ticket` elements in their constructor call. Some of their components may reference specific `Token` and `Infrastructure` objects. If these referenced objects have not been previously instantiated, an illegal configuration exception is thrown.

The simulation is run as a single thread. During each simulation cycle the agents are polled, and provided the opportunity to reevaluate their goals and actions. The loop continues until it has executed a predetermined number of steps or is halted by the user. Goal selection and Action execution are discussed below.

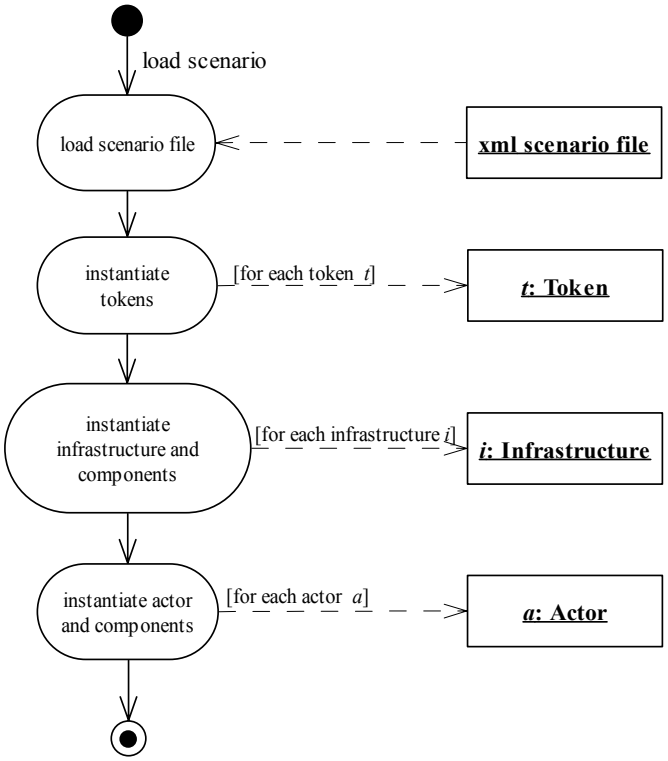


Figure 41. Scenario loading activities for SimManager class.

2. Entity Package

The `entity` package contains the `Entity` class, which is the parent class of all *entities* in the simulation. It also contains two key passive entities that are specializations

of the `Entity` class: `Infrastructure` and `Resource`. These are passive entities indicating that they cannot initiate actions; rather they react to actions initiated by other entities. Active entities are discussed in the `actor` package.

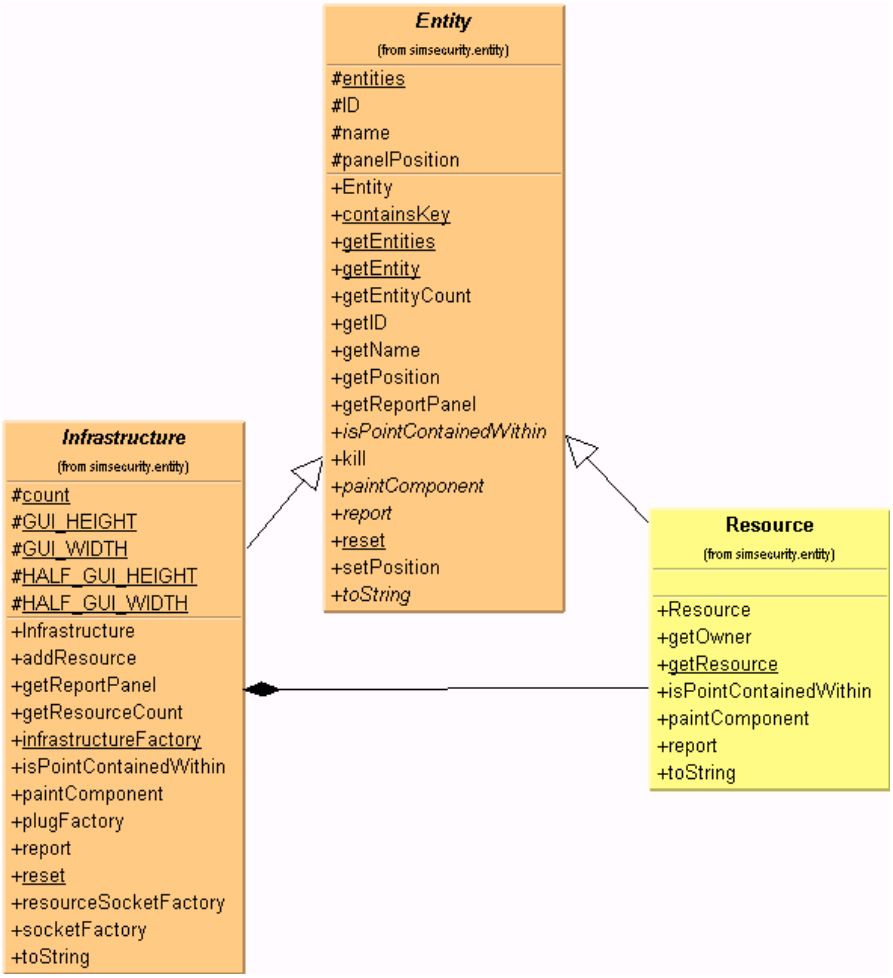


Figure 42. The `entity` package contains the `Entity` class and two specialized passive entities: `Infrastructure` and `Resource`.

3. Actor Package

The `actor` package contains the *active entities* and their component classes. An active entity is an entity that the `SimManager` permits initiating actions. The abstract `Actor` class is the parent of all active entities. The `CompositeAgent` class extends `Actor`, and therefore implements the abstract methods of Actor.

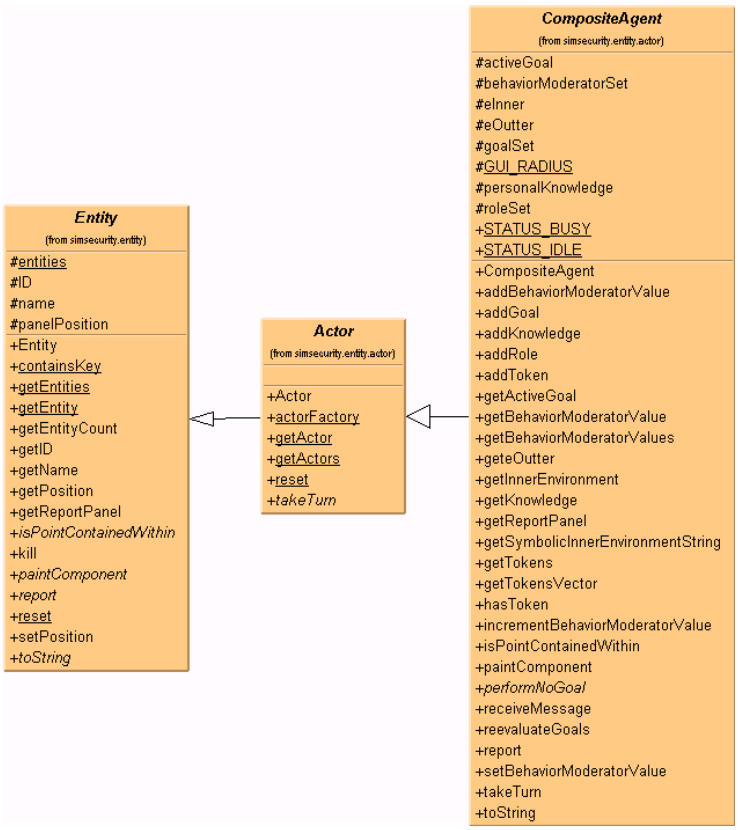


Figure 43. The main classes in the actor package are the `Actor` and `CompositeAgent` classes, which inherit from the `Entity` class.

When an Actor's `takeTurn()` method is called by the `SimManager`, the actor executes its goal selection routine, depicted in Figure 44. Each actor loops through its goal set G and determines which goal to execute next. If the next goal to execute is not the current executing goal, then the current goal is interrupted, resulting in that goal's `onInterupt()` actions being executed.

If no goal has been selected then the agent's `performNoGoal()` method is executed, ensuring any management functions are handled on the Agent. If a goal has been selected for execution then the goal executes its `onExecute()` actions, and the goal is recorded as the current `activeGoal`.

Researchers may customize their own agent architecture and implement it via STIAM by extending the abstract `Actor` class and registering them with `SimManager`. `SimManager` alerts an `Actor` that it may initiate actions by calling its

`EntitytakeTurn()` method. An agent must also provide a `toString()` method and a `report()` method to report the agent's status as text. The `paintComponent()` method is called by the `SimManager` to force the agent to paint itself on the GUI. The `isPointContainedWithin()` method returns a Boolean value indicating if the Point passed as an argument is contained within the `Entity` on the GUI display. The `getReportPanel`() returns a JPanel object representing a report of the current status of the actor, which is displayed in the simulation if a user clicks on the actor on the GUI as indicated by the `isPointContainedWithin`() method.

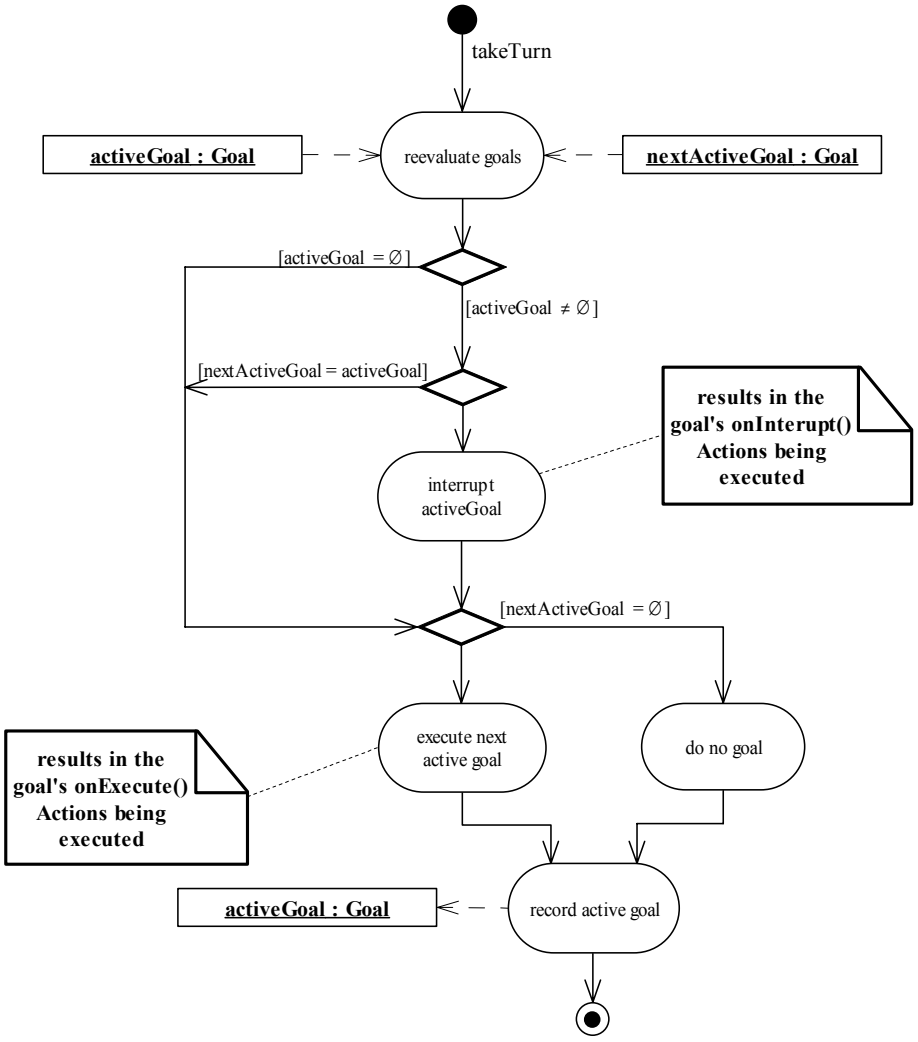


Figure 44. An activity diagram representing an agent goal selection routing.

Figure 45 depicts the algorithm an Agent uses to reevaluate its goals. The Agent loops through each of the Goals that are currently evaluated to be *critical*. The agent selects the critical goal with the greatest weight that has an *active* ticket.

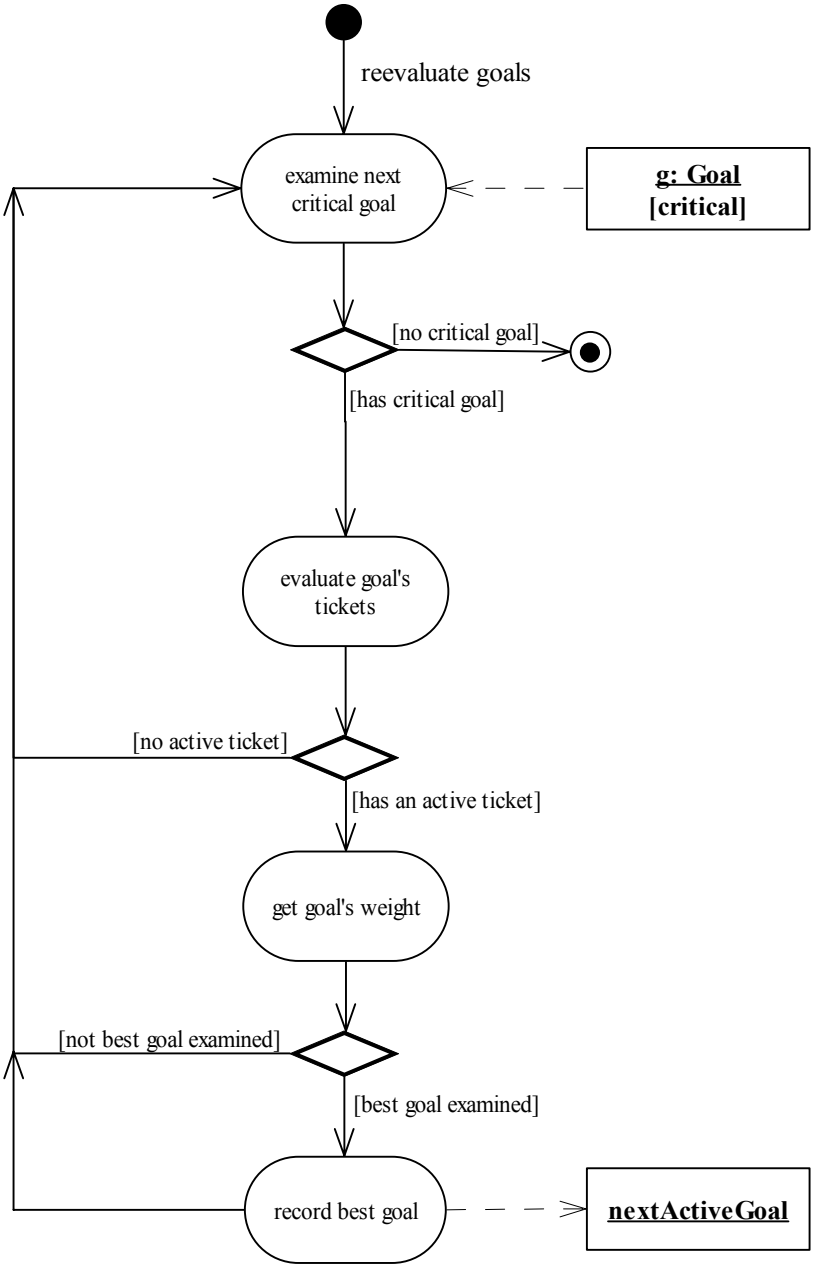


Figure 45. An activity diagram depicting the agent reevaluate goal routine.

Figure 46 illustrates the relationship between roles, actors, goals, and tickets. `CompositeAgents` are assigned to roles. Roles provide the assigned agent with a set of

goals. Goals have tickets that are its procedural problem-solving step. The frames within a ticket may be filled with any object that extends the `Frameable` abstract class: sub tickets, actions, or slots. Slots contain connectors (not depicted) that allow it to dynamically bind to appropriate knowledge pool items at runtime. Complex problem solving can be implemented through creative use of tickets. Illustrated in Figure 46 is the base `Ticket` class. This class is extended to a `SequentialTicket` that always starts in the first frame and executes the frames sequentially, until the last frame is executed, where it is flagged as *completed*. The `ContinualLoopSequentialTicket` class extends the `SequentialTicket` class, overriding the `onCompletion()` method, causing the ticket to loop back to the first frame and continuing indefinitely.

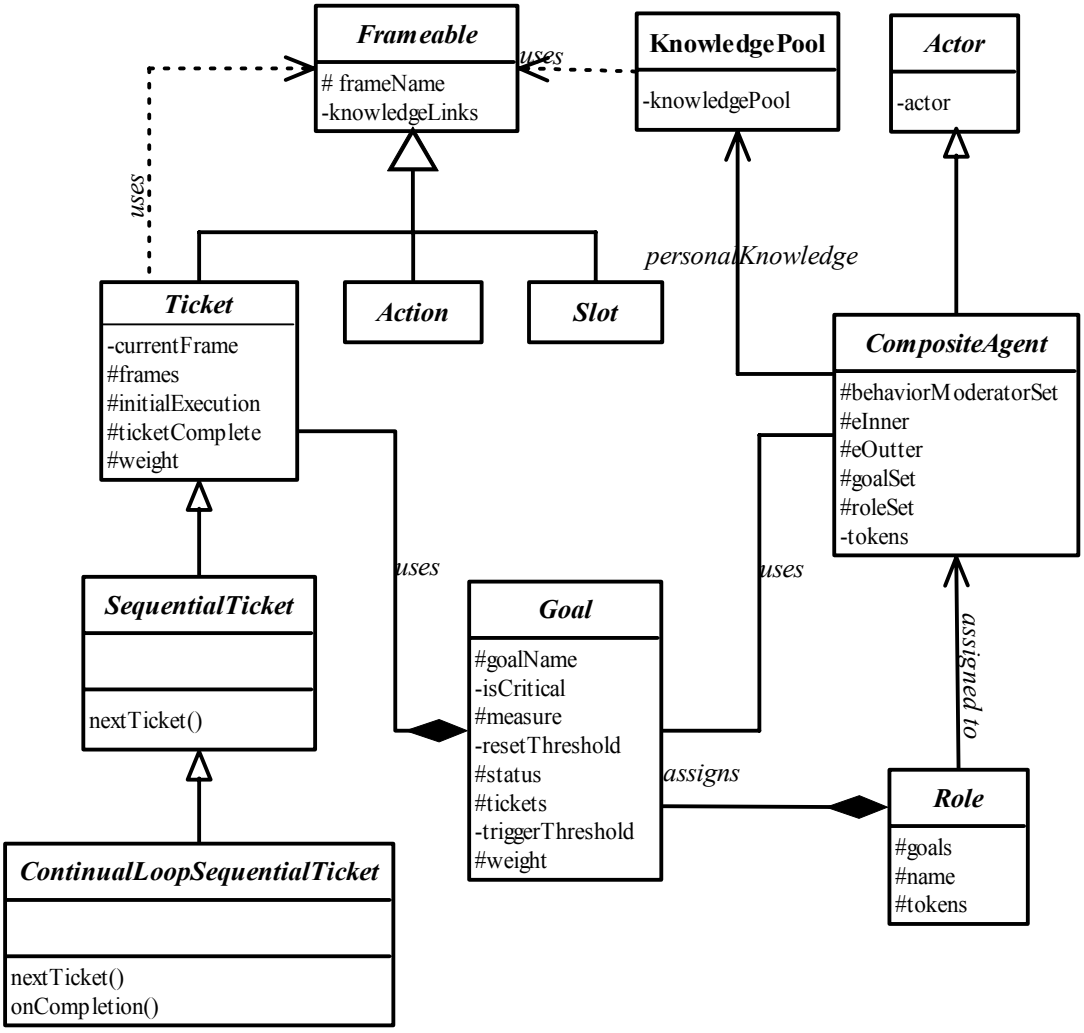


Figure 46. The classes of the `actor` package.

4. Connector Package

The connector package contains all of the necessary classes for implementing `Connector` and `IConnector` systems.

a. IConnectors and IBinders

The `IConnector` is discussed in detail in Chapter IV. In this implementation both the `Socket` and `Plug` classes are extended to include `Connector`, `Listener`, and `Resource` subclasses, where:

- `Connector`: only executes `Actions` on other `IConnector`, and is not directly related to a `Resource`.
- `Listener`: passively reports on existing matching `IConnectors`.
- `Resource`: are linked to a particular `Resource` within an `Infrastructure` and represents an *interface* to a particular `Resource`.

Each `IConnector` is able to designate specific `IConnectorAction` objects to execute upon the `IConnector` connecting, disconnecting, or being interupted. An `IConnectorAction` object has the potential to modify an *infrastructure's* interface.

In this implementation, SimManager has a single instance of IBinder, which acts as an outer environment, and handles all inter-agent communication between IConnector objects.

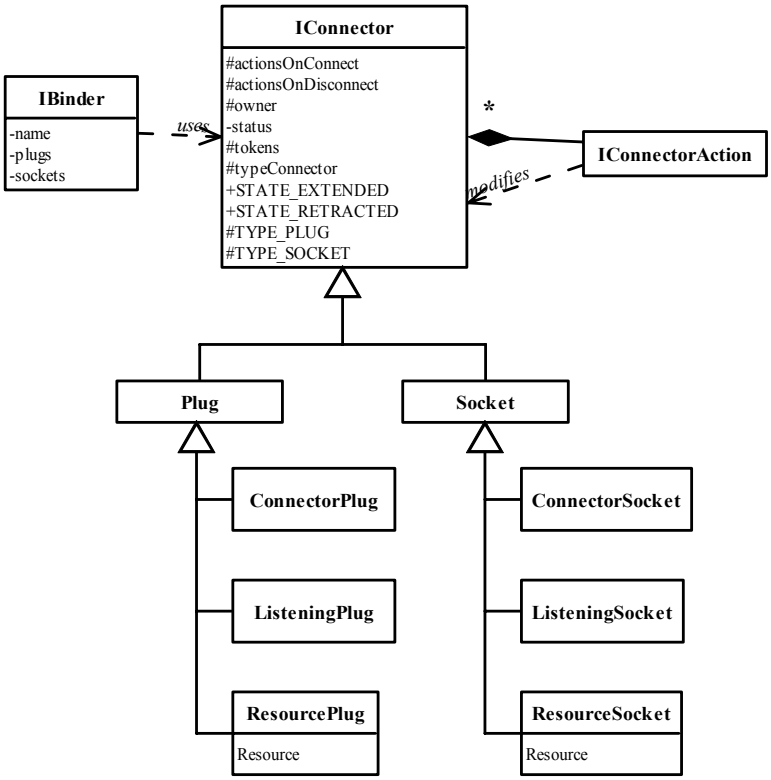


Figure 47. The classes related to `IConnectors` in the `connector` package.

b. Connectors and Binders

Each `CompositeAgent` has a single `Binder` object that represents the actor's inner environment. `Connector` objects are implemented using the same architecture as Java event listeners [Horstmann and Cornell, 2000], and was first implemented using this technique by [Osborn, 2002]. `CompositeAgent` component objects implement the `ConnectorChangeListener` interface, and register with the `Binder` any `Connectors` that they are interested. `Connectors` also register themselves with the `Binder`. The `Binder` notifies registered components of matching `Connectors` when the connector extends into the `Binder`, change value or change state. The notification occurs through a `ConnectorChangeEvent` which is passes to the `ConnectorChangeListener` using the `connectorChanged`() method. The `ConnectorChangeEvent` object contains a reference to the `Connector` that is signaling the component. Connectors are discussed in Chapter V.

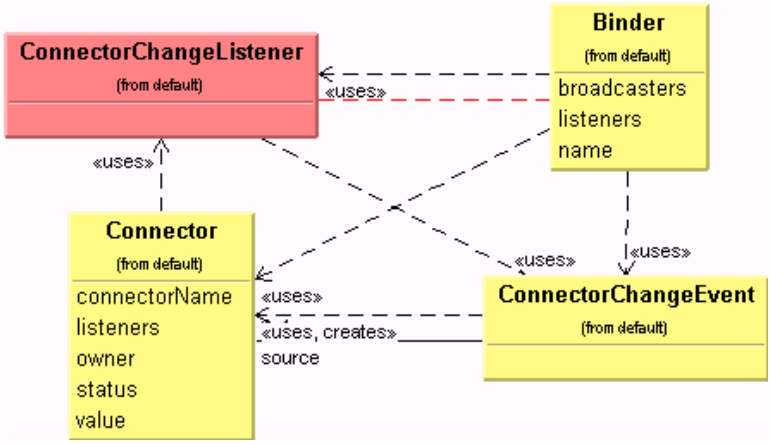


Figure 48. The classes related to `Connectors` in the `connector` package.

5. Scenarios Package

The `scenarios` package contains specific scenario files in XML. Figure 49 depicts a sample XML scenario file. Entities defined in a scenario file are specializations of base entity classes that are placed in this package, permitting `SimManager` to find them at runtime and dynamically bind to the classes.

```
<!—

Actor names must start with capital letter and must match
the class  name exactly.
The actor and infrastructure .class files must exist in the
simsecurity/scenarios/ directory
-->

<scenario>
<tokens>
    <token> dbPassword </token>
    <token> officeAccess </token>
    <token> malice </token>
    <token> vulnerability103 </token>
    <token> systemPatch103 </token>
</tokens>
<infrastructure>
    <class> EnterpriseInfrastructure </class>
    <name> enterprise </name>
</infrastructure>
<actor>
    <class> UserCompositeAgent </class>
    <name> user1 </name>
</actor>
<actor>
    <class> UserCompositeAgent </class>
    <name> user2 </name>
</actor>
<actor>
    <class> UserCompositeAgent </class>
    <name> user3 </name>
</actor>
<actor>
    <class> HackerCompositeAgent </class>
    <name> hacker1 </name>
</actor>
</scenario>
```

Figure 49. Sample XML scenario file.

6. Utilities Package

The `utilities` package contains the basic utilities needed for the simulation. This includes a `clock` that maintains the current simulation cycle. `Assert` is a singleton debug class that has a single method with two arguments. `Assert` prints an error message to the *standard output* and halts the application if the first argument passed in the method does not evaluate to `true`. `ReadScenario` and `LoadClasses` are used by

the `SimManager` for reading the XML scenario files and loading classes dynamically at runtime. `BucketHashtable` and `ConnectorHashtable` are all specialized data structures and algorithms for dealing with connectors and tickets.

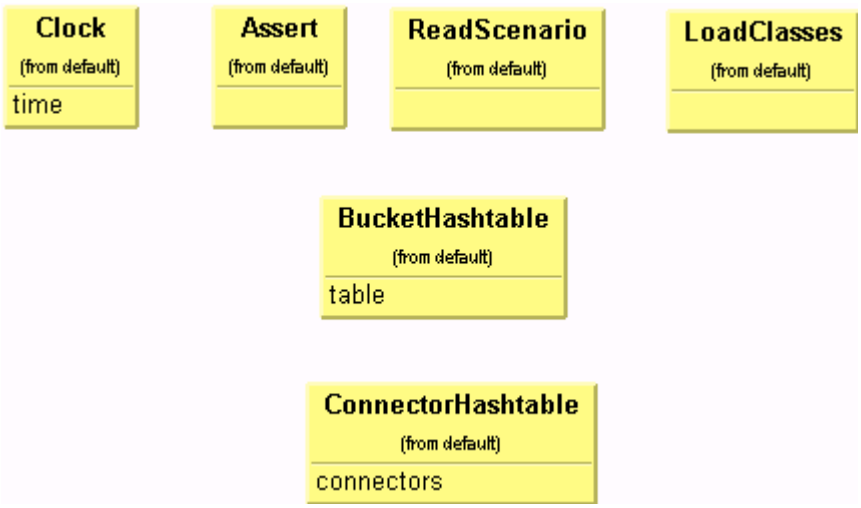


Figure 50. Classes in the `utilities` package.

C. SUMMARY

This implementation of STIAM was as a single Java application running a single simulation thread. The actors were polled, giving each an opportunity to execute actions. The architecture proved to be modular and robust, facilitating the testing of new components and ideas without extensive modification of existing code.

The following chapters introduce several scenarios that were developed using this software, and the results obtained from their implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. SCENARIO IMPLEMENTATION

A. INTRODUCTION

This chapter introduces several scenarios that were developed using the software implemented in this dissertation. The first scenario is a demonstration of an attacker actor who adapts to the environment and discovers successful attack sequences that are not encoded in the agent. The second scenario models system exploit propagation, and illustrates how STIAM can be used as a virtual laboratory to illustrate complex IA domains. Combined, these scenarios provide an introduction into some of the IA scenarios that are possible using this implementation of the STIAM model. By comparing the data obtained in the later scenario with observations in the IA field, a validation of the hypothesis generation capability of STIAM is provided.

B. SCENARIO ONE – "ADAPTIVE ATTACKER"

The implementation of this scenario has several purposes:

- to provide a proof of principle of basic model elements,
- to show that an actor is able to discover an attack sequence that it was not previously aware when the simulation started,
- to demonstrate how attackers adapt.

1. Background

The society of this scenario contains one actor and three infrastructures. The single actor is called Hacker[14]. The Hacker goals are to increase knowledge of computer systems, expand access to systems, and earn fame within the hacker community. The infrastructures consist of a data library, a hacker community, and an enterprise infrastructure. The library is a simple repository of information, which responds to queries, providing data to whomever requests it. The hacker community is a repository of system vulnerabilities, providing the means to access identified systems. The

[14] The term Hacker, as used in this chapter, refers to "a malicious or inquisitive meddler who tries to discover information by poking around … possibly by deceptive or illegal means…" [Steele *et al.*, 1983].

enterprise is a corporate infrastructure, consisting of a critical resource called database, and numerous infrastructure interfaces. One interface permits users to access the database resource. Other interfaces represent means to gather information about the corporation, and vulnerabilities that exist on the infrastructure.

Assumptions: A clock cycle represents an arbitrary unit of time (though fairly short) corresponding to the duration of simulation events.

2. Implementation

The Society is defined as five types of *tokens*, three *infrastructures*, and one *actor*.

a. Tokens

The tokens in this scenario represent the knowledge that an actor has, or needs, to achieve his goals. The tokens are:

Token Name	Initially Possessed By	Description
dbPassword	enterprise	This token represents the current password needed to access the corporate database.
enterprise	hacker	This token represents the identity of a particular corporate enterprise.
vuln103	hackSite	An actor possessing this token has the knowledge required to exploit a technical vulnerability numbered 103.
enterpriseService	enterprise	This token represents general-purpose information on computer processes and services that may be operating on an infrastructure.
sysType	enterprise	This token represents technical information about the information technology operating in the infrastructure, and accessible through an interface.

Table 4. The Tokens used in Scenario One.

b. The Infrastructures

There are three infrastructures. The first is depicted in Figure 51. It depicts a critical resource labeled ‘database’ that is accessible for read access if an entity presents an database plug with the token ‘dbPassword.’ This represents the ability to access a corporate information resource via a password.

A second interface is accessible via an enterprise plug with token ‘enterpriseService.’ This interface represents the ability of an entity to see what

information services are operating on an infrastructure. In real networks, this represents a means to identify what services or processes are running on nodes in a network. To simplify the model, this scenario only models one service or process, which is called 'systype', and knowledge of this service is represented by the token 'systype.' Thus, binding to this enterprise socket results in the entity that caused this binding to receive a token labeled 'systype', indicating that the entity now "knows the type of system running on the infrastructure."

The last interface on the enterprise infrastructure is a system vulnerability. Binding to this vulnerability socket requires an entity to posses token 'vuln103', i.e. knowledge of the exploit for this particular vulnerability. Successfully binding results in the binding entity receiving the password to the resource, depicted as the 'dbPassword' token.

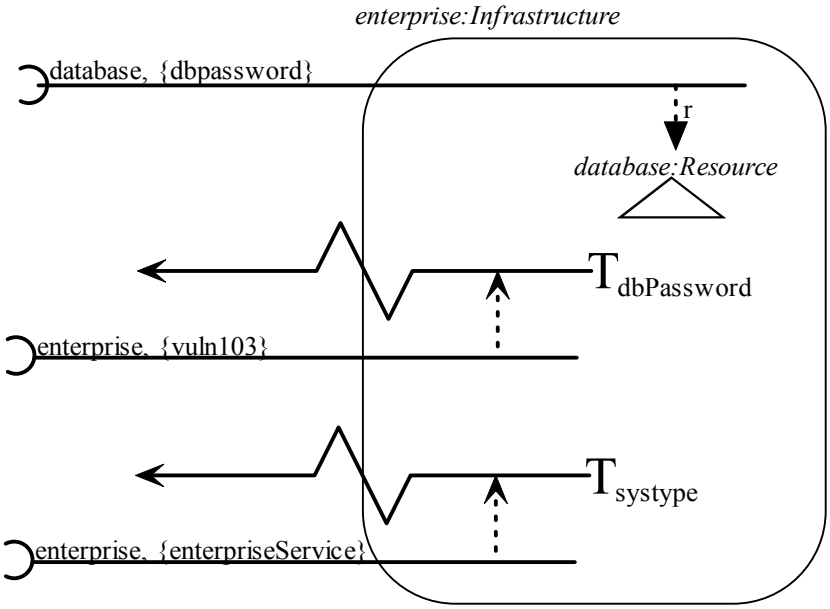


Figure 51. An `enterprise` *infrastructure*, with a *resource*, service scan, and vulnerability

The second infrastructure, depicted in Figure 52, represents a traditional library, a source of publicly available, open-source information. The library has one interface and simple functionality. Binding to the interface is performed through the library socket that requires a single token called 'enterprise.' Successfully binding to this

interface results in a token labeled ‘enterpriseService’ being transmitted to the entity on the other end of the plug. This capability represent the ability of any person going to a public source, presenting the name of the entity, and receive public information on the enterprise.

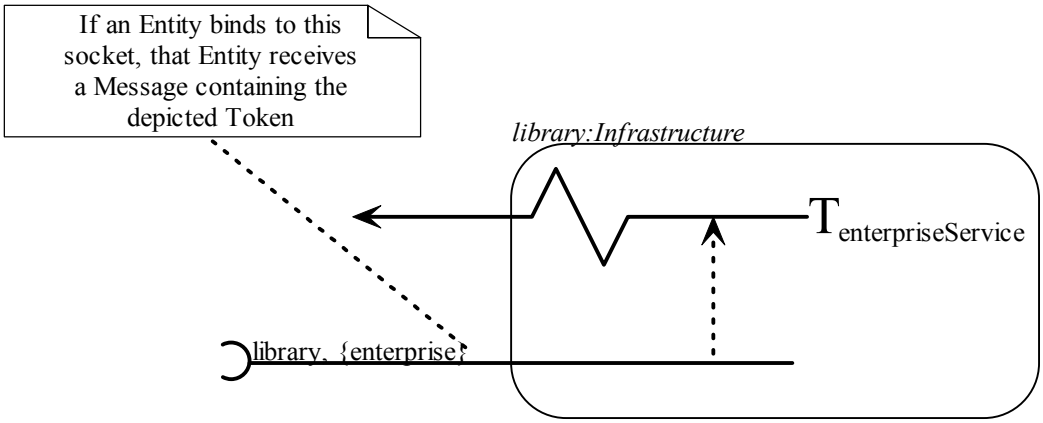


Figure 52. The `library` infrastructure, which provides information on the `enterprise` infrastructure.

The hackerSite infrastructure is similar to the library, and is depicted in Figure 53. This infrastructure represents a *very* simplified version of the hacker community, and its vulnerability-exploit sharing process. In this infrastructure, an entity may present system information and receive, in the form of a token, an exploit for the particular system.

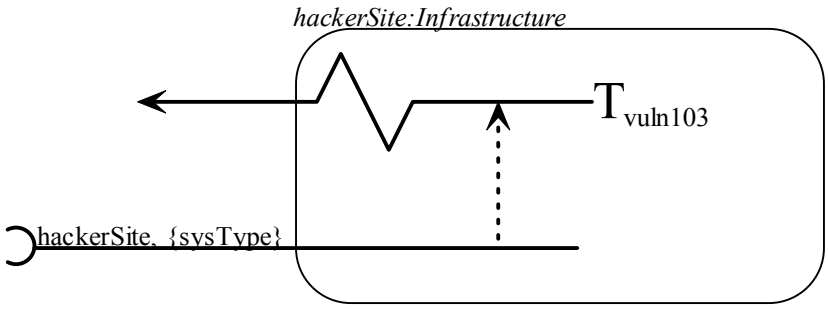


Figure 53. The hackerSite infrastructure, which provides vuln103 Token if presented with sysType Token.

c. Actors

There is one actor class, called Hacker. The hacker begins with only one token, the `enterprise` token, indicating that the hacker has a designated target, the `enterprise`. The hacker's goal components are illustrated in Figure 54. This class diagram shows that a hacker has three goals;

1. Gather intelligence: this goal is for the hacker to gather information on targets that it believes are important. To achieve this goal the hacker has three processes:
    a. Conduct library research: the hacker presents tokens to the library in the hopes that the hacker will receive important public information.
    b. Scan an Enterprise infrastructure: the hacker scans an infrastructure, and attempts to retrieve any system data on from the infrastructure.
    c. Research System Vulnerabilities: the hacker takes any system information it may receive and presents it to the hacker community in the hopes of receiving exploits against the particular systems.
2. Expand personal powerbase: The hacker takes exploits against a particular system and executes the exploits, achieving access to the inner workings of the penetrated infrastructure.
3. Earn fame: The hacker takes critical infrastructure information and the ability to penetrate an infrastructure, and accesses the critical resources of the infrastructure.

The goals have an implied priority based on their weights. This indicates that it is more important to earn fame then expand the power base, or gather intelligence. This is by no means a comprehensive hacker goal structure, so additional goals, tickets, and actions can be added to hacker actors to explore their implications.

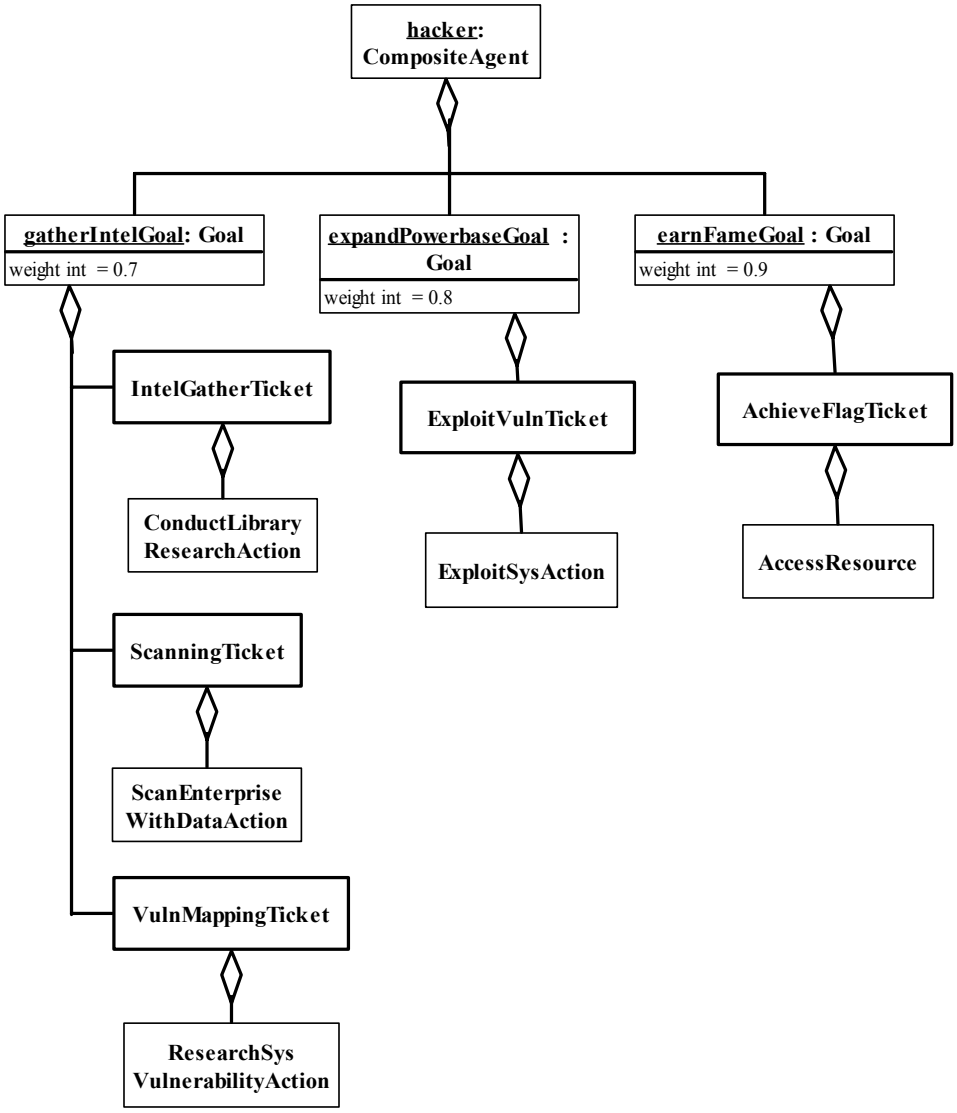


Figure 54. The example hacker's goals, tickets and actions.

Figure 55 presents a screenshot of this scenario in execution. In this implementation of STIAM, the actors are always depicted as circles, vertically along the left edge of the screen. Infrastructures are depicted as ovals along the right edge. Resources are depicted as triangles contained within their respective infrastructures. Figure 55 depicts an actor, labeled "hacker" bound to a socket of the "enterprise" infrastructure.

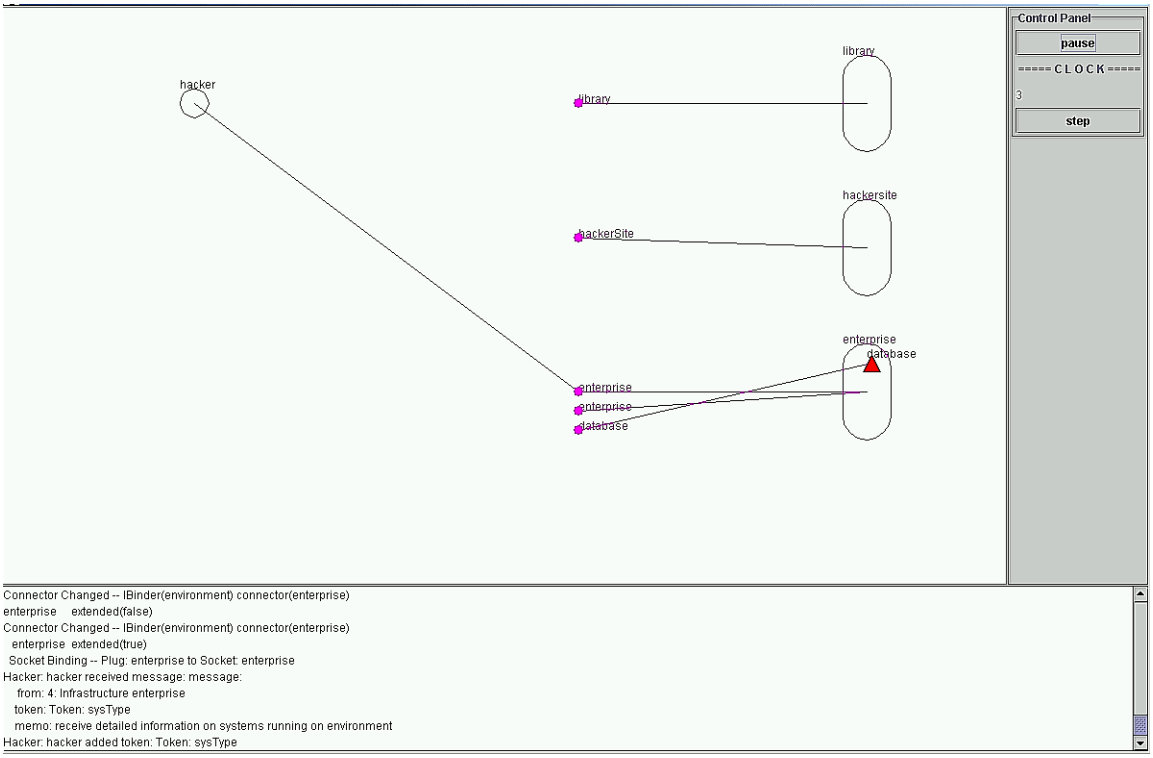


Figure 55. Screen shot of Scenario One on STIAM implementation.

3. Experimental results of Scenario One

a. Observations

The sequence of key actions that occur in this scenario are listed in Table 5. Appendix A contains the complete list of actions that occurred in this scenario.

The scenario begins with the actor in possession of only one token; the `enterprise` token. The actor attempts to solve its highest priority goal that has an action that it can perform.

Clock Cycle	Output from simulation	Explanation

1	`Hacker: hacker executing goal Goal: GatherIntelGoal`
	`executing frame: Action: ConductLibraryResearchAction`
	`Socket Binding -- hacker to library`
	`Hacker: hacker received message: message:`
	`from: 0: Infrastructure library`
	`token: Token: enterpriseService`
	`memo: receive information on enterprise`
	`Hacker: hacker added token: Token: enterpriseService`
		The hacker accesses the library and requests information on `enterprise`, which it received in the form of a new `enterpriseService` token.

2	`Hacker: hacker executing goal Goal: GatherIntelGoal` `executing frame: Action: ConductLibraryResearchAction`	The hacker tries the new `enterpriseService` token at the library, which fails

3	`Hacker: hacker executing goal Goal: GatherIntelGoal`
	`executing frame: Action: ScanEnterpriseWithDataAction`
	`Socket Binding -- hacker to enterprise`
	`Hacker: hacker received message: message:`
	`from: 4: Infrastructure enterprise`
	`token: Token: sysType`
	`memo: receive detailed information on systems running on environment`
	`Hacker: hacker added token: Token: sysType`
		The hacker tries the `enterpriseService` token against the `enterprise` infrastructure, which is successful, returning a new `sysType` token.

4	`Hacker: hacker executing goal Goal: GatherIntelGoal` `executing frame: Action: ConductLibraryResearchAction`	The hacker tries the `sysType` token at the library, which fails.
5	`Hacker: hacker executing goal Goal: GatherIntelGoal` `executing frame: Action: ScanEnterpriseWithDataAction`	The hacker tries the `sysType` token at the `enterprise` infrastructure, which fails.

6	`Hacker: hacker executing goal Goal: GatherIntelGoal`
	`executing frame: Action: ResearchSysVulnAction`
	`Socket Binding -- hacker to hackersite`
	`Hacker: hacker received message: message:`
	`from: 2: Infrastructure hackersite`
	`token: Token: vuln103`
	`memo: receive exploit for 'vuln103' on system 'systype'`
	`Hacker: hacker added token: Token: vuln103`
		The hacker tries the `sysType` token at the `hackerSite`, which is successful, resulting in receiving a new `vuln103` token

7	`Hacker: hacker executing goal Goal: ExpandPowerbaseGoal`
	`executing frame: Action: ExploitSysAction`
	`Hacker: hacker received message: message:`
	`from: 4: Infrastructure enterprise`
	`token: Token: dbPassword`
	`memo: receive password to access Resource:database`
	`Hacker: hacker added token: Token: dbPassword`
		The hacker uses the `vuln103` token on the `enterprise` infrastructure, which results in the hacker receiving a new token `dbPassword`

8	`Hacker: hacker executing goal Goal: EarnFameGoal`
	`executing frame: Action: AccessResourceAction`
	`** success **`
		Hacker uses the dbPassword token on the enterprise infrastructure and successfully access the critical resource.

Table 5. Sequence of steps used by `Hacker` to access the critical resource.

b. Discussion

While this scenario may appears simple, it is important to understand that the hacker actor does not possess an explicit plan on how to access the 'database' resource prior to the run of the simulation. Nor does the attacker generate a plan prior to the execution of an action. The Actor fired the highest priority goals whose prerequisites were met, and discovered the sequence of steps that led to accessing the resource. The actor sensed the environment that it was presented, and used its limited abilities to discover what works.

Not all of the hacker's actions were successful. In clock intervals 2,4 and 5 the hacker presented iconnectors to infrastructures that failed to bind. These unsuccessful actions were the results of the actor attempting to solve the `GatherIntelGoal` by presenting new information tokens that it had received, to any intelligence source. Cycle 2 and 4 represent the hackers unsuccessfully research of information at a library, and cycle 5 represents unsuccessfully research at the enterprise.

In cycle 5 the hacker presented information tokens to the infrastructure that 'obviously' will not produce any results. In the real world, security analysts say this is an indication of a *script kiddie*, or unskilled attacker. Script Kiddies may try anything in an attempt to access information system, without understanding the underlying technology [The Honeypot Project, 2002].

c. Lessons Learned

The actors presented in this dissertation generate plans through reactive interactions within the environment that they are placed. This can be contrasted with the method traditional rule-based systems use to generate their plans.

On can think of traditional rule-based systems as starting at the root of a search tree and generating the tree[15]. A node on the tree represents a state, and a transition on the tree represents a subgoal or action taken by the agent. The leaves of the

[15] This represents a forward chaining search. An agent could also perform a backward chaining search where it starts at the goal and generates the tree back toward the current state [Russell and Norvig, 1995].

tree represent goal states. The shape of the tree is specified and constrained by the rules contained within the agent. Once the tree has been generated, the agent selects an action to perform that leads the agent down the tree to the desired goal. In many domains, generating the entire tree may be intractable, so the agent must stop at some point and select an action to perform that appears promising.

In this dissertation, an agent does not perform a search for a goal, nor generate a plan tree. Rather, it selects the highest goal that has an action ready for execution and executes the action. During the execution of the simulation, an implied search tree is created by the actions the agent selects and the goals that are achieved. This can be thought of as dynamically generating implied plans during runtime.

The tree generated by the agent during runtime does have human bias. A static weight is applied to the goals and tickets when they are input into the agent. These weights act as a heuristic, aiding the agent in achieving goals. The heuristics act as a means to prune the search tree as the agent runs through the simulation.

The advantage of this reactive planning is that the agent is able to deal with unspecified environments. Additionally, there is no time-consuming search, which is beneficial to real-time system. Additionally, with the addition of weight adjustment, an agent could discover what works, and what doesn't work, in never before seen environments, and adjust the weight of tickets appropriately. This is left for future work.

There are several disadvantage of the reactive planning. First, the agent may suffer from the effects of linear problem solving as discussed in Chapter V. Second, the agent only has a local perspective, possibly resulting in the "horizon problem" [Russell and Norvig, 1995] where the agent commits to a path based on a local perspective, leading to a future unavoidable failure. This problem may be solved by a look-ahead planning algorithm, but this would fundamentally change the behavior of the actor, and may prove detrimental to the innovative reactive plans desired.

C. SCENARIO TWO – WINDOW OF VULNERABILITY

This scenario examines information system exploit propagation. An information system flaw is an unspecified functionality on a particular information system that results from poor system design, implementation, or maintenance [Myers, 1980]. Once a *flaw* has been discovered, and there exists a potential to exploit the flaw causing undesirable consequences on the part of the defender, then the flaw becomes a *vulnerability*. When a flaw has been identified, a vendor may provide a *patch* or other means to remove or mitigate the flaw or the effects of the flaw. A vulnerability or exploit may also be *publicized* resulting in a rapid propagation of exploits throughout the society. In addition, the vulnerability may become *scripted*, so that less sophisticated attackers, script kiddies, may exploit the more complex vulnerabilities without sophisticated technical knowledge. This sequence of actions is called the "window of vulnerability" [Arbaugh *et al.*, 2000].

This section discusses the window of vulnerability. It presents a model that generates the sequence of actions in a virtual society. The results of this scenario are compared with the results obtained by Arbaugh *et al.* [2000]. This section validates the claim that STIAM can produce hypothesis that are comparable to what is observed in the IA environment.

1. Background

To model a widely distributed vulnerability, a larger and more complex society was created.

The society contains seventeen actors:

- two sophisticated hackers,
- five script kiddies,
- ten system administrators.

Also, the society contains thirteen infrastructures:

- an elite infrastructure,
- a script kiddie infrastructure,
- a vendor infrastructure,
- ten enterprise infrastructures.

The enterprise infrastructures are a homogeneous set of infrastructures that have the same, single vulnerability. There is one system administrator actor responsible for each enterprise infrastructure. The system administrators have goals of discovering vulnerabilities and exploits against their infrastructures and keeping their infrastructures patched. The vulnerability on each infrastructure will let any entity who possesses a special token, "vuln1" to bind to the infrastructure. This binding is the goal of the attackers, and an action the defenders wish to prevent.

In addition, this scenario models the hacker community. An attacker is modeled as someone who is capable of discovering vulnerabilities and exploits, and distributing knowledge of these to the rest of the hacker community via hacker infrastructures. These infrastructures can be accesses by less sophisticated attackers, called script kiddies, who can obtain these exploits. The script kiddies are then able to exploit vulnerabilities without possessing the technical skill to develop the exploits themselves. The hackers have a higher skill level than script kiddies, but otherwise possess identical goal and action sets.

Finally, we wish to model the vendor community. System administrators report to a vendor when the infrastructure for which the system administrator is responsible has been attacked. The vendor creates a patch, which is published to the society, and may be retrieved as a token by system administrators. The system administrators then install the patch on their system, resulting in the elimination of the appropriate vulnerability.

By varying properties of the infrastructures and actors, a virtual laboratory exists whereby security analysts may examine the results of changes to the society and observe how these changes affect the society. Results obtained from various experiments are presented in a later section.

2. Implementation

The society is defined as four tokens, four types of infrastructures, and three types of actors.

a. Tokens

The tokens in this scenario are:

Token Name	Initially Possessed By	Description
vuln1	none	An actor possessing this token has the knowledge required to exploit the technical vulnerability on the enterprise infrastructure.
patch1	none	An actor possessing this token has the knowledge and tools to patch or mitigate the effects of the vuln1 vulnerability.
notify	sysadmins	This token represents a message from a system administrator to a vendor that the system administrator's infrastructure has been exploited (by vuln1).
sysadmin	enterprise infrastructures	This token is used by the enterprise infrastructure to indicate that a message (iconnector) is designated for a system administrator only. This may represent a trusted communication channel or confidentiality method that is used between the infrastructure and the administrator.

Table 6. The Tokens used in Scenario Two.

b. The Infrastructures

The four types of infrastructures are defined as the *elite*, *script*, and *vendor* infrastructure, and multiple *enterprise* infrastructures.

The elite and script infrastructures are identical in functionality and represent the information system used by criminal attackers. The elite infrastructure accepts messages containing a token from another actor. When the message arrives, it is added to the infrastructure's token set. Additionally a socket is extended that allows any actor to bind who presents a plug iconnector labeled "*elite*". Binding to this socket results in a token being sent by message to the owner of the plug. This socket binding represents the ability of anyone in the hacker elite community to bind to the elite infrastructure to receive any tokens possessed by the infrastructure. Figure 56 (a) represents an elite infrastructure that contains a single token, "*vuln1*". The "script" infrastructure is identical to the "elite" infrastructure except that the socket is labeled as "script" rather than "elite". Exploits are published to elite sites first and later scripts sites, representing the ability of elite hackers to exploit systems earlier than the script kiddies. Figure 56 (b) represents the "*script*" infrastructure, which is identical to the *elite*, except for the socket label.

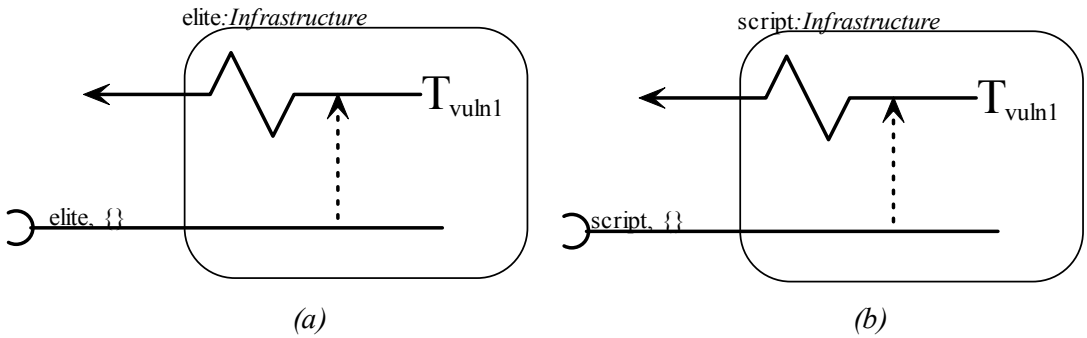


Figure 56. The elite (a) and script (b) infrastructures are identical except for the socket labels.

In this scenario, there are ten nearly identical enterprise infrastructures. The only difference between the instantiations of the infrastructures is the enterprise label on a plug and socket connector. These labels are identifies as $enterprise_n$ where n ranges from 1 to 10, representing the identity of the infrastructure.

An enterprise infrastructure has a vulnerability labeled "vuln" which requires one token, "vuln1", which represents the ability to exploit the vulnerability. Upon exploiting the vulnerability a plug is extended that has the potential to alert the system administrator that the infrastructure was exploited. This represents an abstraction of "after-the fact" alert mechanisms, such as intrusion detection systems. Additionally, a socket labeled "$enterprise_n$, {patch1}" exists that represents the ability of a system administrator to patch the vulnerability. Binding to this socket results in the vulnerability socket retracting permanently.

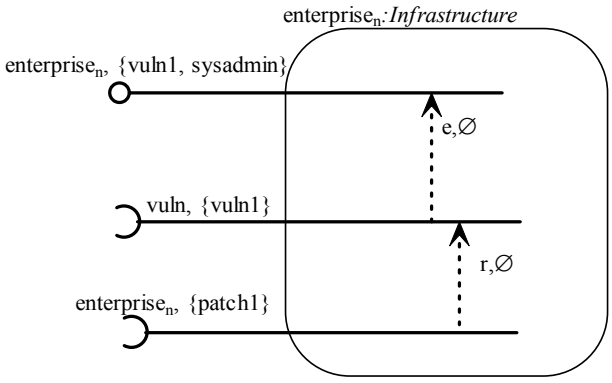


Figure 57. The enterprise infrastructure has an alert plug, a vulnerability socket, and a patch socket.

The vendor infrastructure depicted in Figure 58 represents the entire vendor community. The vendor infrastructure consists of a single active socket. System administrators bind to this socket to notify the vendor that their systems have been exploited. The first actor that binds to this socket causes the extension of another socket representing the availability of a patch for the vulnerability "vuln1". If an entity binds to this second socket, it will receive a message from the vendor containing the *patch1* token, which represents a patch to the vulnerability *vuln1*.

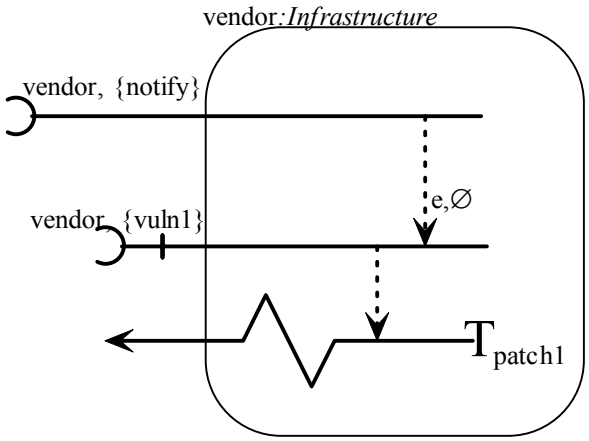


Figure 58. The vendor infrastructure represents the entire vendor community.

c. Actors

As discussed earlier, hackers (elites) and script kiddies (scripts) were modeled identically except for higher skill value provided to elites and their ability to bind to their respective infrastructures. The goal structures are presented below in Figure 59. Each attacker has a goal of acquiring vulnerabilities. If a new exploit is not available on the hacker websites, the attacker will try to generate a new exploit. The probability of an attacker generating an exploit on any simulation cycle is the skill level of the attacker; this value ranges from 0.0 to 1.0 for scripts, and 0.5 to 1.0 for elites.

If an attacker discovers an exploit, in the form of a new token, the attacker will publish the exploit. The first turn with the exploit the attacker will publish the exploit to the elite infrastructure. The attacker will delay for a preset number of turns, then publish the exploit on the script website. This represents the attackers desire to publish exploits to the elite site first, in order to gain fame within the hacker community.

While the attacker is waiting to publish on the script site, and after it publishes on the script site, the attacker will use the exploit against infrastructures. This represents the hacker using exploits it has discovered against targets of interest.

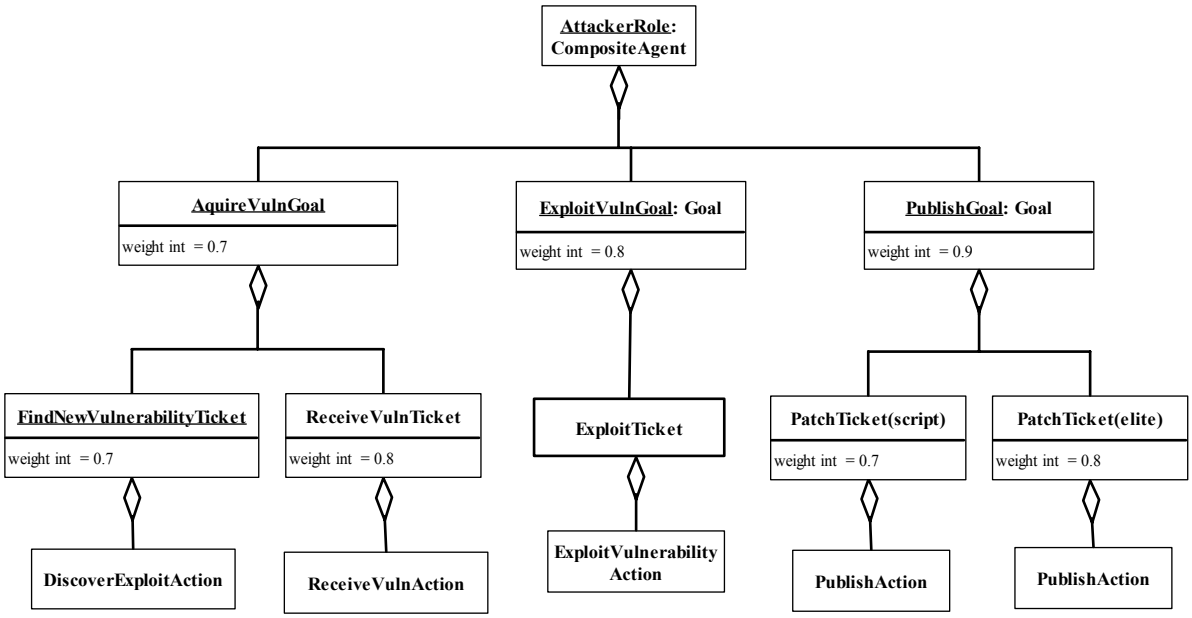


Figure 59. The Attacker role consists of three goals: acquire, exploit, and publish vulnerabilities.

The system administrator role, as depicted in Figure 60, has two goals; discover any exploits occurring on the system it is assigned, and patch exploits that are discovered. The system administrator has a socket extended from the `DiscoverExploitAction`. When the actor's infrastructure is exploited, the infrastructure binds to this socket, notifying the actor of the exploit. This action is then marked as completed, and the next frame in the `DiscoverExploitTicket` executes, notifying the vendor through an iconnector that the system administrator's infrastructure has been exploited.

During each turn, the `GetPatchAction` extends a plug, awaiting notification of new patches from the vendor. If a vendor has extended a socket advertising a new patch, the `GetPatchAction` binds, resulting in the actor receiving the patch through a message. The existence of this new patch causes the

`ApplyPatchAction` to fire the next turn, resulting in the actor patching the vulnerability on its infrastructure.

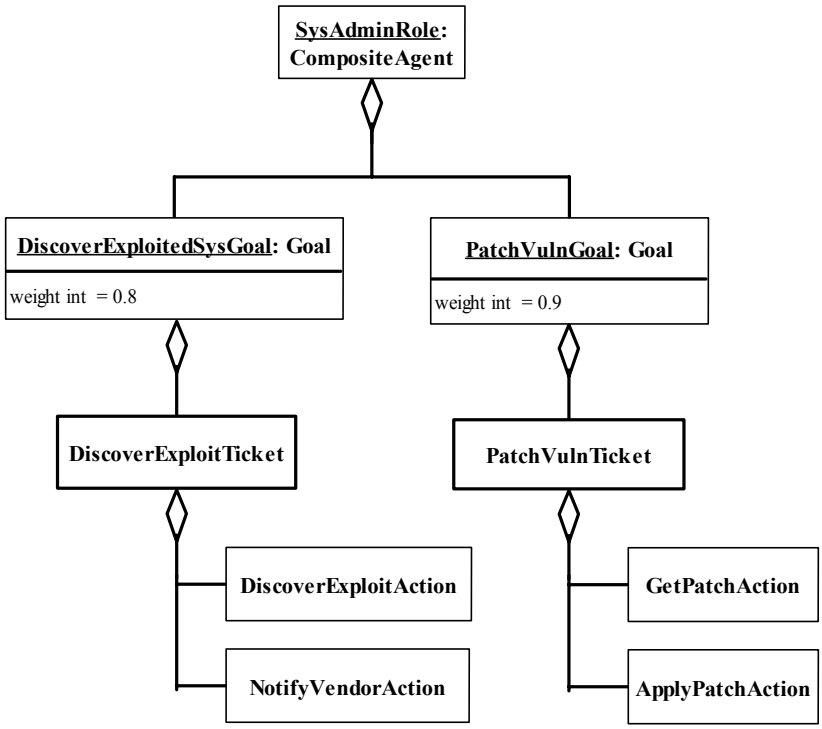


Figure 60. The system administrator role.

A security analyst can model proactive system administrators versus reactive system administrators very easily. In its current configuration, the system administrator actor will bind to the vendor and patch its infrastructure as soon as a patch is available, representing a proactive system administrator. This may be misleading, since this implementation of the system administrator does not have any other conflicting goals. To cause the system administrator to be reactive, an internal connector labels "exploited" is placed in the system administrator actor. The `GetPatchAction` can then be forbidden to execute until the internal connector extends. The connector will be extended by the `DiscoverExploitAction` after detection of the infrastructure being exploited.

Alternatively, a reactive systems administrator could be modeled with a single goal and a single sequential RespondTicket. This ticket would have discover, notify, receive patch, and apply patch actions placed in a sequential order.

Future work could include more goals that conflict with a system administrator's priorities, such as upgrading systems, fixing user problems, and general system maintenance. This is left as future work.

Figure 61 is a screen shot of the Window of Vulnerability scenario at the start of execution. The hackers and script kiddies are represented vertically along the left edge. The system administrators under the attackers, labeled "enterprise1" through "enterprise10", represent the system administrators for the similarly labeled infrastructures. Sockets are represented as hollow circle connector ends, and plugs are filled circle connector ends. Retracted connectors are to the right of the infrastructures. All other connectors are extended.

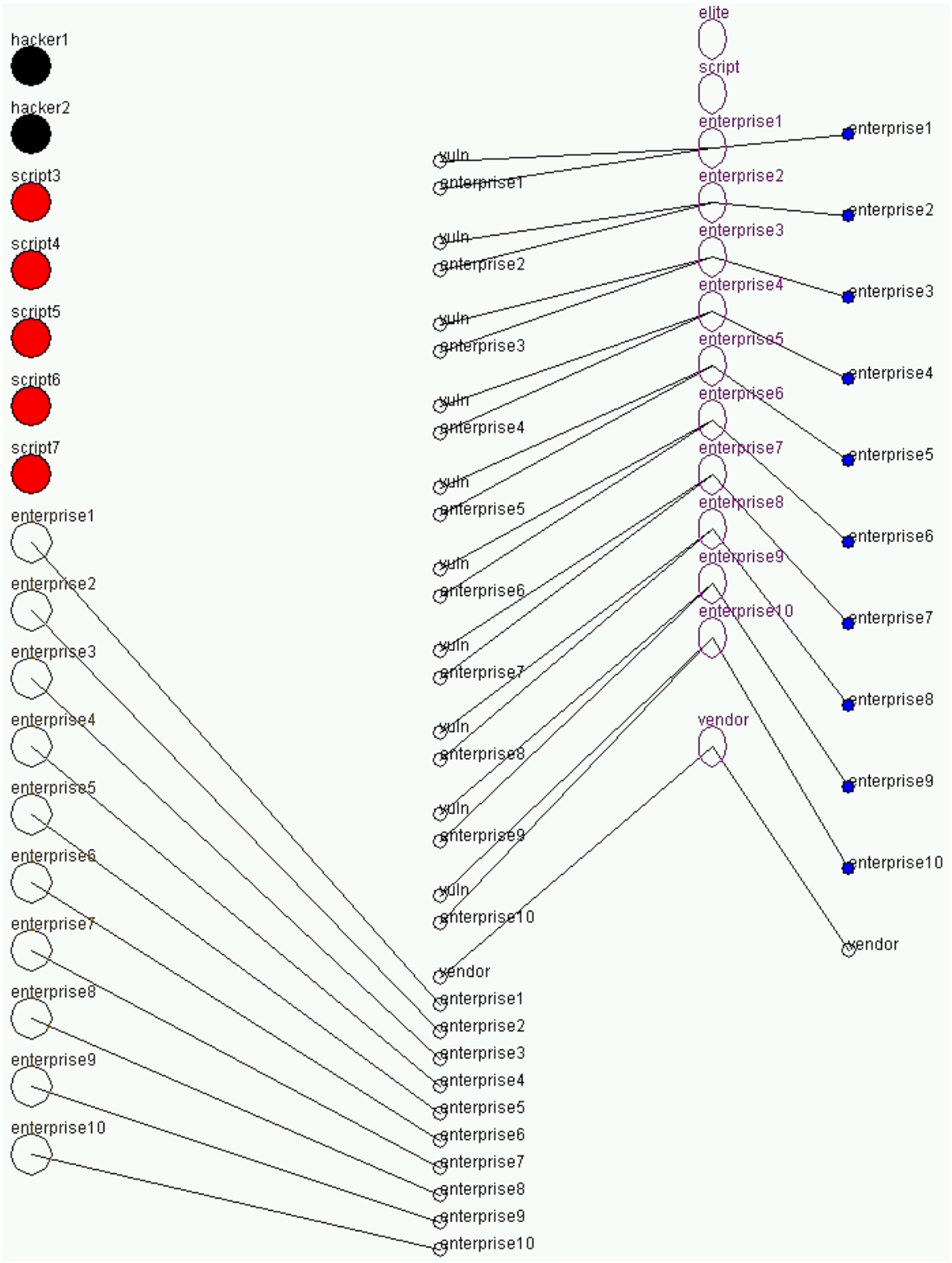


Figure 61. Implementation of Scenario Two.

3. Experimental results of Scenario Two

Arbaugh *et al.* [2000] provided the first quantitative analysis of the window of opportunity phenomena. They discovered that the number of incidents reported to the Computer Emergency Response Team Crisis Center (CERT/CC) Incident Team, when plotted over time, is positively skewed toward the beginning of the vulnerability reporting, as depicted in Figure 62. Arbaugh *et al.* discovered that after an exploit is discovered there is a small increase in the number of exploit incidents. This is followed by a tremendous jump in the number of exploit incidents as the exploits are published to the general community and scripts are developed for the exploits. After patches are released, the number of exploits begins to drop, and continues to drop slowly, sometimes over a period of years, as system administrators apply patches to their systems [Arbaugh *et al.*, 2000].

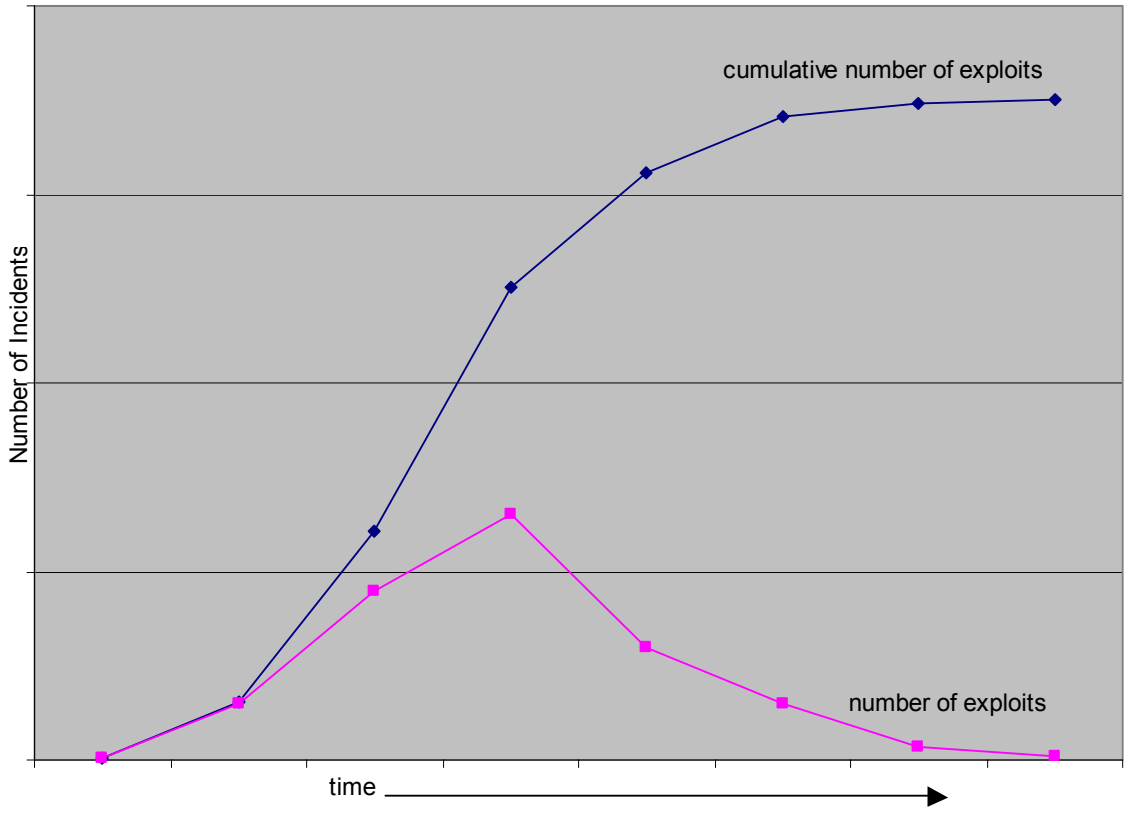


Figure 62. Typical exploit distribution graph shape reported by Arbaugh et al., 2000.

Furthermore, Arbaugh *et al.* discovered that patches were normally available simultaneously to, or shortly after public disclosure of a vulnerability, i.e. before the largest number of reported exploits. This can be attributed to the system's administrators being unaware of, or not responsive to, installing patches and other mitigating means to infrastructures [Arbaugh *et al.*, 2000]. The real catalyst for the increased of incidents observed is the scripting of exploits. Scripting involves creating a tool that requires very little technical skill, typically facilitating an unsophisticated attacker to use more sophisticated means.

Arbaugh *et al.* acknowledges that the data used in their analysis was not complete, and pointed out several reasons for this inadequacy. STIAM, therefore, can provide a virtual laboratory to perform hypothesis generation, and to adjust system and actor parameters to observe the effects to the IA of an organization, or the society as a whole.

a. Observations

Seven separate sub-scenarios were implemented and analyzed. The results of these simulation runs are presented.

In the first run, the system administrator agents were encoded to react to their system being attacked. These agents represent reactive system administrators who do not apply patches until after their systems have been attacked. Figure 63 depicts the results of this scenario. The character labels on the graph represent:

a) Initial exploit – this represents the first time an attack is executed on a system.
b) Publication of exploit on elite site – this is the first indication that the exploit has been published, but only to a limited community.
c) Publication of exploit on script site – at this point the exploit has been scripted and has a widespread distribution.
d) Publication of patch – here a vendor publishes a patch for the vulnerability and systems should begin being secured.

Reactively applying patches results in an exploit existing in the society for a longer period. Attackers are able to exploit the initial infrastructures, followed by the system administrators of these infrastructures discover that their systems are compromised and begin the process of acquiring and installing patches. As these initial

infrastructures are patched, the attackers move on to other infrastructures that have not been patched. This sequence continues until all of the infrastructures have been attacked and subsequently patched. The result is that all (100%) of the infrastructures are exploited

The constant slope of the cumulative line from turn 9 to turn 19 in Figure 63 is caused by the constant number of attackers successfully exploiting infrastructures during those turns. Additionally, the simulation provides perfect situational awareness, resulting in an analyst receiving all of the reports for all exploits, something not possible in a real environment.

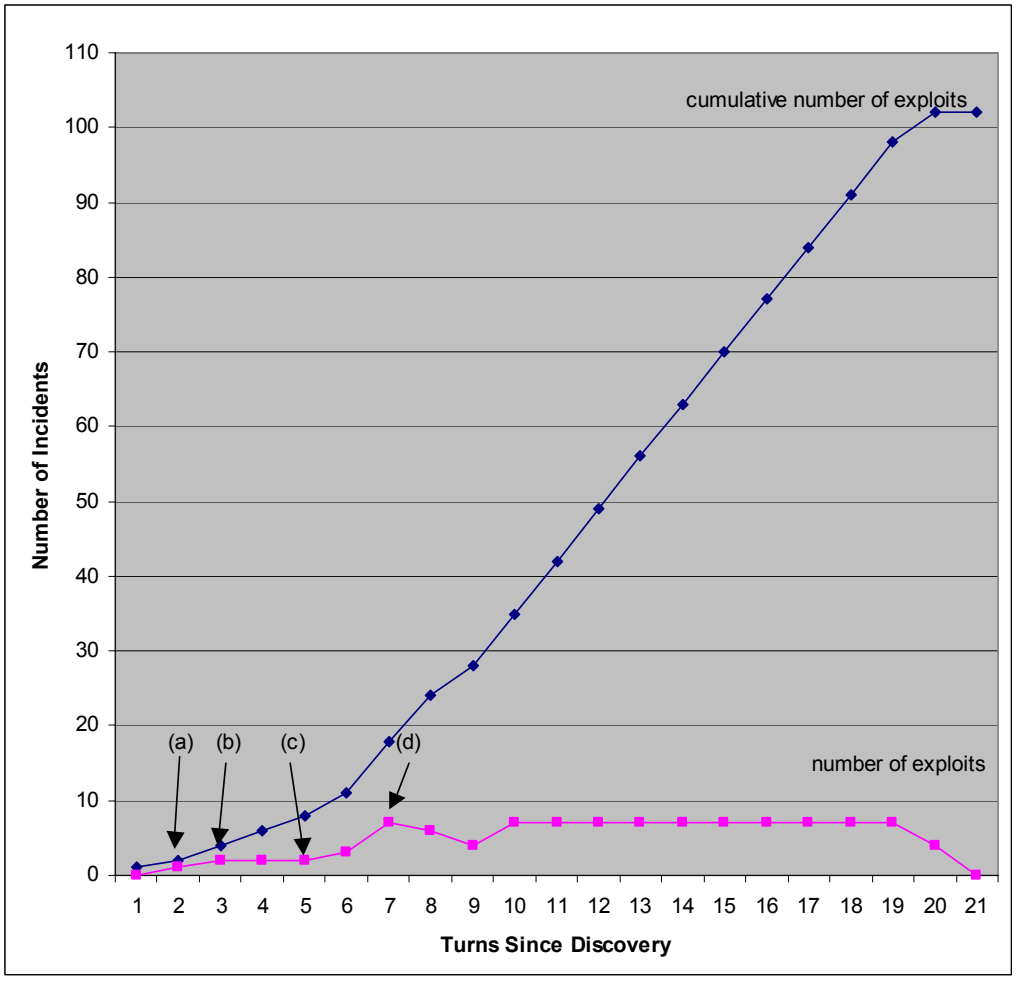


Figure 63. Results of reactive system administrators with patch released after scripts.

The next example uses reactive system administrators also. Here, the script is released with the elite exploit. There is a very rapid rise in the number of attacks (c). The number of attacks stays constant even after the patch (d) is published, because the attackers simply jump to an infrastructure that has not been attacked, and therefore not patched. There is a slight decrease in the number of incidents and the lifetime of the exploit because the attackers are able to exploit systems quickly at the beginning of the lifeline, and therefore the systems administrators get a chance to patch their systems quicker. The result is still 100% of the infrastructures penetrated.

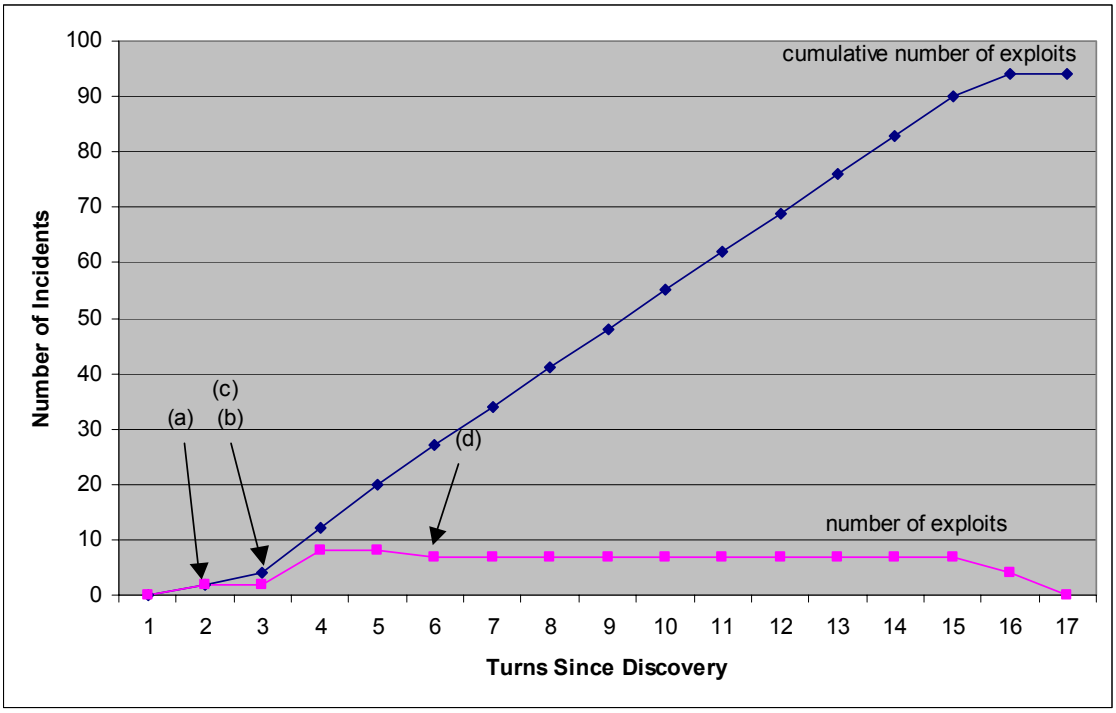


Figure 64. Reactive system administrator with accelerated publication of script and delayed publication of patch.

In the next example the reactive system administrators have access to the patch before the script is released. Seven infrastructures are quickly exploited, and the system administrators rapidly respond, patching their systems. As systems are patched, other systems are exploited, causing an oscillation in the number of exploits over time. Eventually all systems are exploited, and then patched, and the vulnerability dies.

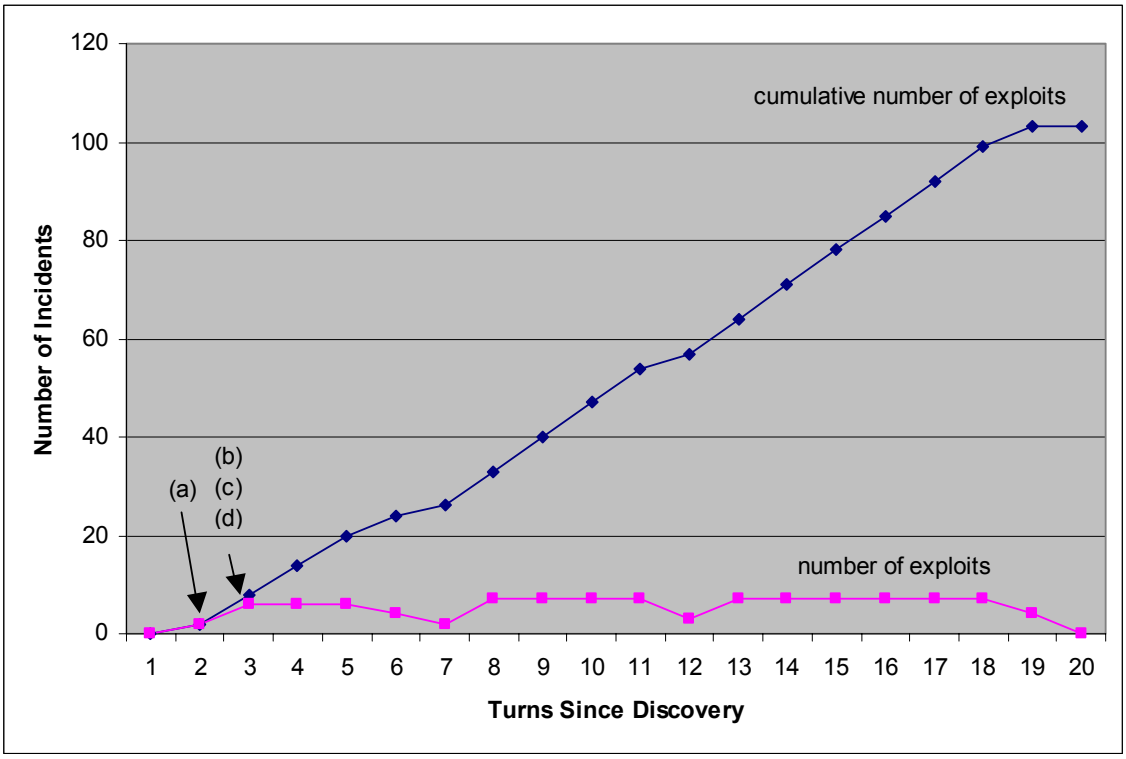


Figure 65. Reactive system administrator with patch released prior to scripts.

In the next run, the system administrators apply patches as soon as a patch is available from a vendor, representing proactive system administrators. The results are depicted in Figure 66. As shown, an attacker discovers the exploit at (a). There is a small increase in exploits, representing the hackers receiving the exploit after it is posted to the elite site, and then attacking the infrastructures (b). Next, the script kiddies receive the exploit after it is posted to the script website, which causes a large increase in the number of exploits (c). After a delay the vendor releases the patch (d), all system administrators, whether their systems have been attacked or not, request and apply the patch, whereby the number of exploits drops to zero.

In Figure 66 the vulnerability dies very quickly when compared to reactive scenarios. The result is that far fewer incidents occur, and only half of the infrastructures were compromised. Although the vendor has a long delay in getting the patch to the system administrators, the initiative of the system administrators quickly makes up for this delay by securing both exploited and vulnerable but nonexploited systems.

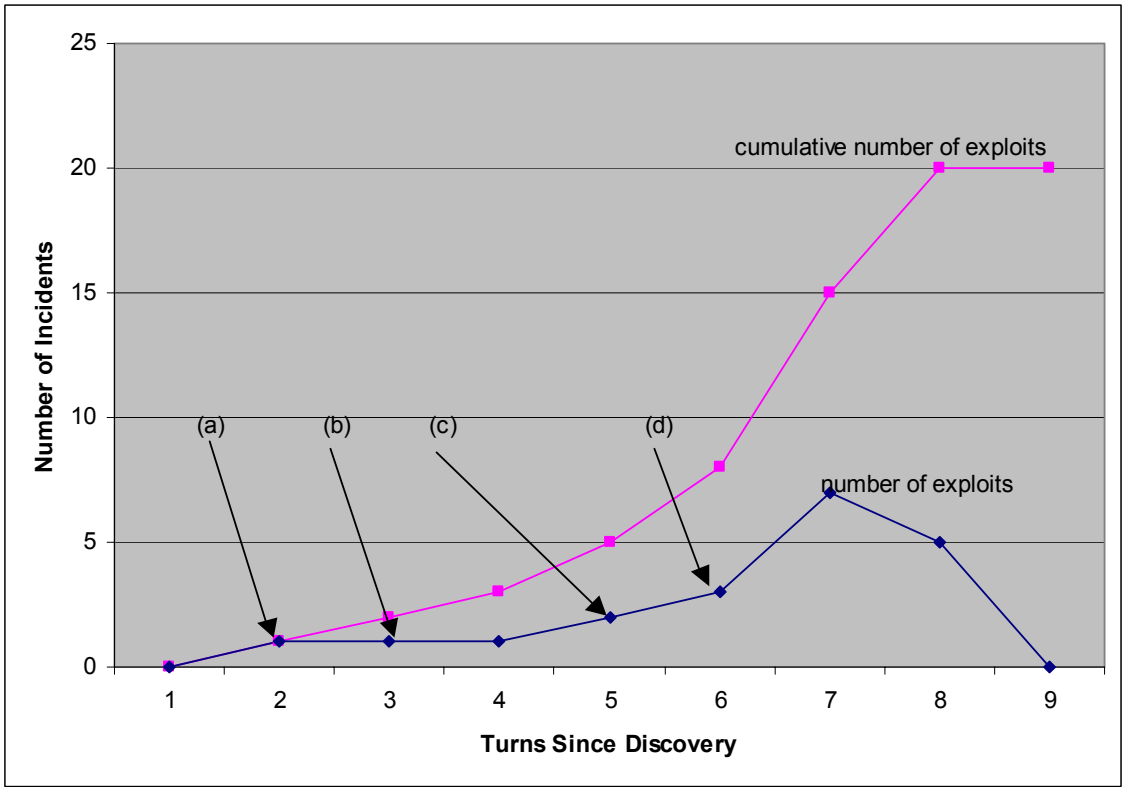


Figure 66. Results with proactive system administrators with patch released after scripts.

In the next run, the exploit is released to elites and scripts simultaneously (b)(c), as depicted in Figure 67. The patch is published after the scripts (d). The result is that, although the lifetime of the vulnerability is identical to the previous scenario, there is a 50% increase in the number of incidents, and all of the infrastructures are exploited, representing a 100% increase. This scenario illustrates that delaying the release of scripts may have a significant impact on the number of systems exploited.

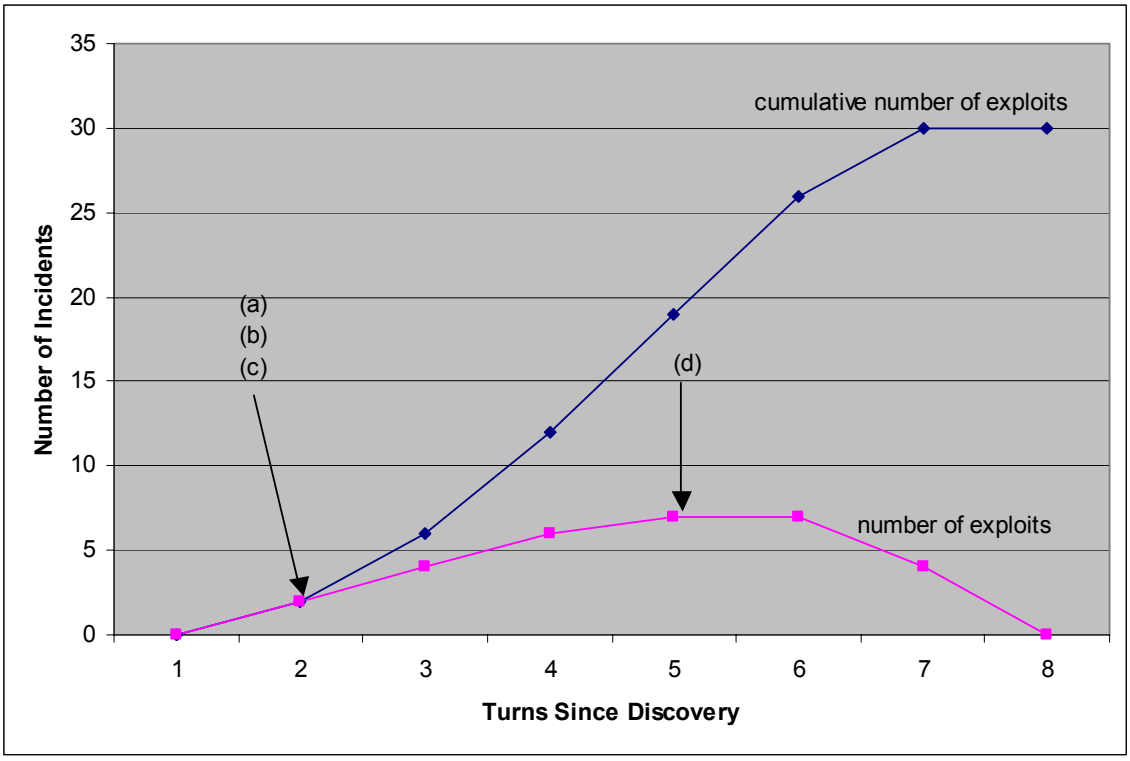


Figure 67. Proactive system administrators with scripts released soon after the elites and the before the patch.

In this next example, Figure 68, proactive system administrators receive the patch for a vulnerability prior to publication of the exploit to script kiddies. This represents a vendor's ability to quickly create a patch once a vulnerability is discovered, or the security communities willingness to wait for a patch to be published before publication of exploits to the script kiddies and the general public.

The number of exploits per turn reaches a maximum of two. This is due to the fact that the exploit was published to the elites, and every elite could then exploit the systems. Once the patch was released, the vulnerability dies before the scripts could exploit systems.

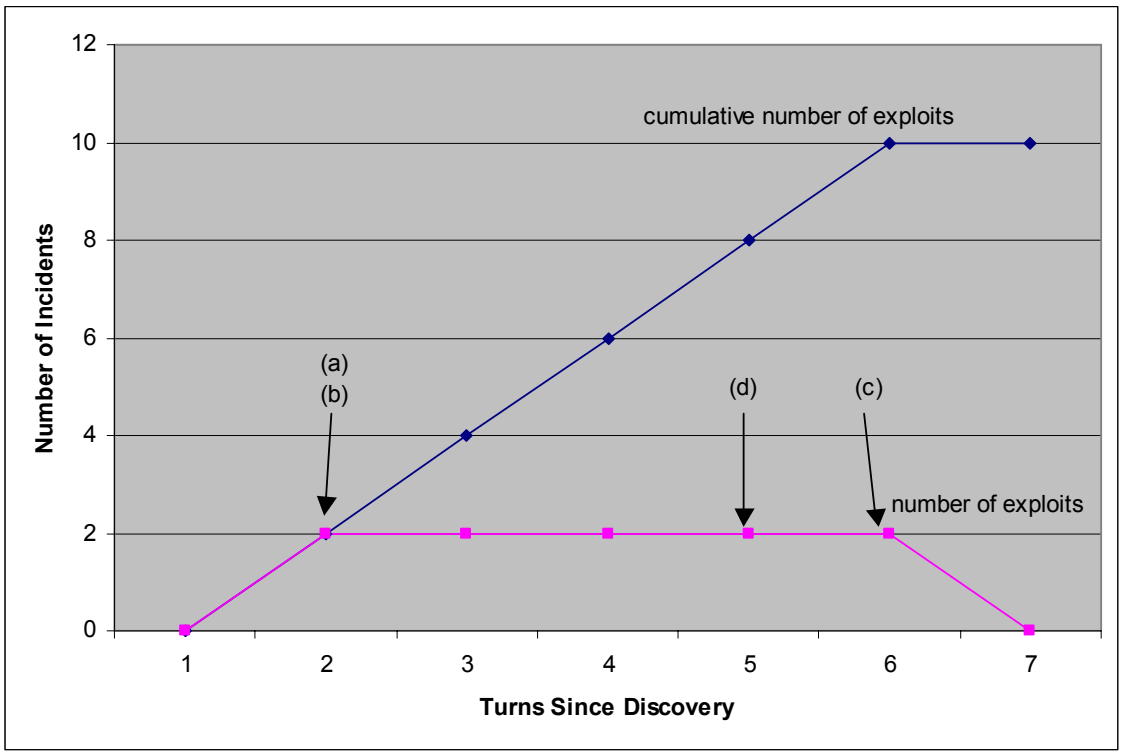


Figure 68. Patch released prior to publication of exploit on script kiddie infrastructure for proactive system administrators.

In this last example the number of script kiddies is increased from five to eleven, giving a total of 13 attackers. In this example the attackers "out number" the infrastructures, so once the exploit is published there is a large number of exploits recorded. The systems are rapidly patched.

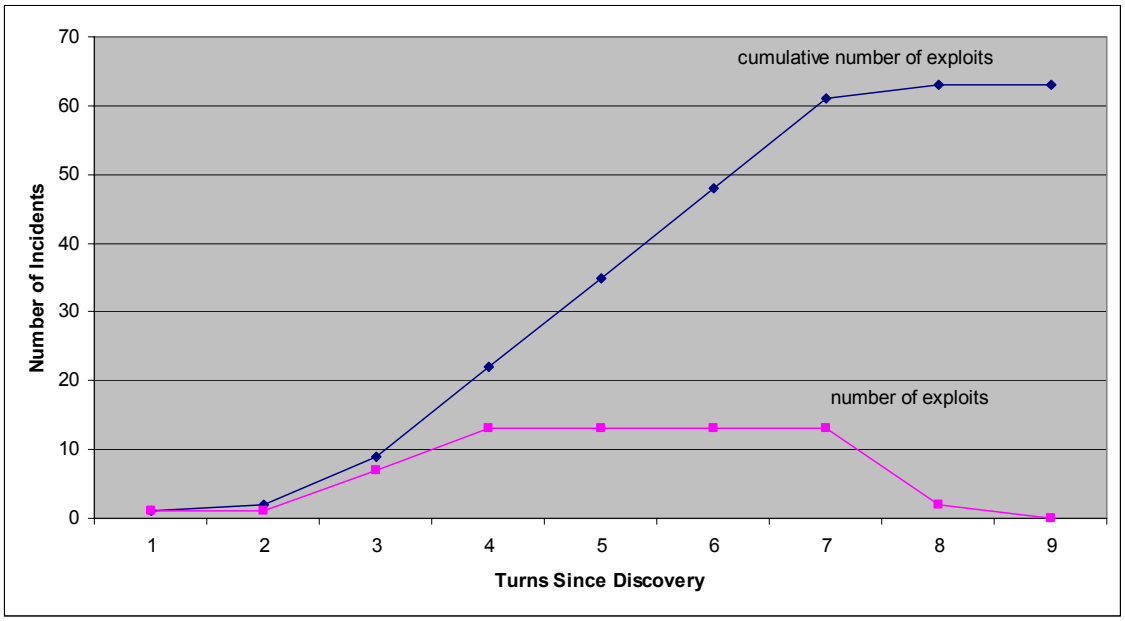


Figure 69. Society with large number of attackers than infrastructures; using reactive system administrators.

b. Discussion

The results of the scenarios presented in the section above are summarized in Table 7.

The first observation, and generated hypothesis, is intuitively obvious. *The largest contribution to securing systems in this scenario is for the system administrator to be proactive in "hardening" their systems, before they are attacked*. When the systems administrator agent reacted to being attacked, it was already too late. First, the reactive scenarios resulted in all of the systems being exploited, an obvious conclusion. Second, the system administrator agents needed to acquire the patch, and install the patch, which took time. This reaction time in installing the patch resulted in attackers being inside a system for long periods of time. Once the system was hardened, the attackers moved on. The result is that the vulnerability timeline were very long for reactive system administrator scenarios.

The second observation, and generated hypothesis, is that *patched need to published prior to the publication of vulnerabilities and scripts to the general public*.

Being proactive in installing patched is of no use if a large number of attackers have a means into a system before a patch is available.

As supported by empirical results, "automation (of attacks) is the catalyst for wide spread intrusions" [Arbaugh *et al.*, 2000]. Delaying the distribution of these automated exploits until a patch is published results in far fewer incidents.

Title of Scenario	Total Number of Incidents	Lifetime of Vulnerability	Number of Systems Exploited
Reactive Sysadmin patch released after script	102	21	100% (10/10)
Reactive Sysadmin script and elites released together patch released after script	94	17	100% (10/10)
Reactive Sysadmin patch released before script	103	20	100% (10/10)
Proactive Sysadmin patch released after script	20	8	50% 5/10)
Proactive Sysadmin script and elites released together patch released after script	30	8	100% (10/10)
Proactive Sysadmin patch released before script	10	7	40% (4/10)

Table 7. Results of Window of Vulnerability Scenario

Numerous factors were not included in this proof-of-principle scenario, and are left as future work. Some of these factors are:

- Collaboration among attackers who "know" each other and can share discovered exploits without releasing them to the general public.
- Competition among attackers, so hackers will install a "backdoor" on compromised system, and then patch the vulnerability that allowed the attacker into the system. The backdoor will provide later access to the system, and patching the system will deny the system to other hackers.
- System administrators may shut down a system if it is compromised. The system may stay "offline" until a patch exists and the system is returned to a safe state. The result should be a shorter lifeline for an exploit, since attackers

would simply move on to other systems when a compromised system is no longer available.

- Conflicting priorities for a system administrator's time, such as maintaining systems, and responding to other actor's demands. An example of conflicting priorities is the system administrator's desire to shut off a compromised system versus a user's desire to perform work activities on the system.

If a system administrator doesn't report an attack against its systems, the vendor may be delayed in being notified and start working on the patch. A way around this is for the vendors to monitor the hacker sites, and begin working on the patch as soon as the exploits are posted. This technique works only if vendors covertly monitor hacker sites, and allegedly occurs within the security community. The hacker groups try to defeat this technique by limiting membership and authenticating potential members [Taylor, 1999].

c. Lessons Learned

Often, we wish to model the effect of time on the simulation. Time is not a component of the basic STIAM model. To consider the effects of time, a delay was placed on the extension and retraction of connectors. For example, when a systems administrator agent binds to the vendors agent to notify the vendor of an exploit, there could be a delay imposed on the extension of the patch distribution sockets. This issue is included in the future work section of Chapter IX.

D. OBSERVATIONS

1. Model Granularity

The decision to model individual actors and infrastructures, or aggregate them into relatively homogeneous entities, depends on the desires of the researcher. Modeling an organization as a single individual and a single infrastructure is relatively simple, but may provide limited insight. Large organizations can be modeled as sets of smaller sub organizations and their respective infrastructures to provide more detailed results, but this adds to the simulation complexity.

To increase the granularity of a model and simulation:

1. deaggregate the organization into important sub organizations, including appropriate roles.
2. deaggregate an infrastructure into separate infrastructures.
3. add additional tokens as appropriate to infrastructure connectors and roles
4. add additional connectors for the interfaces required.

Figure 70 depicts a single organization and infrastructure that was deaggregated into three organizations and three infrastructures.

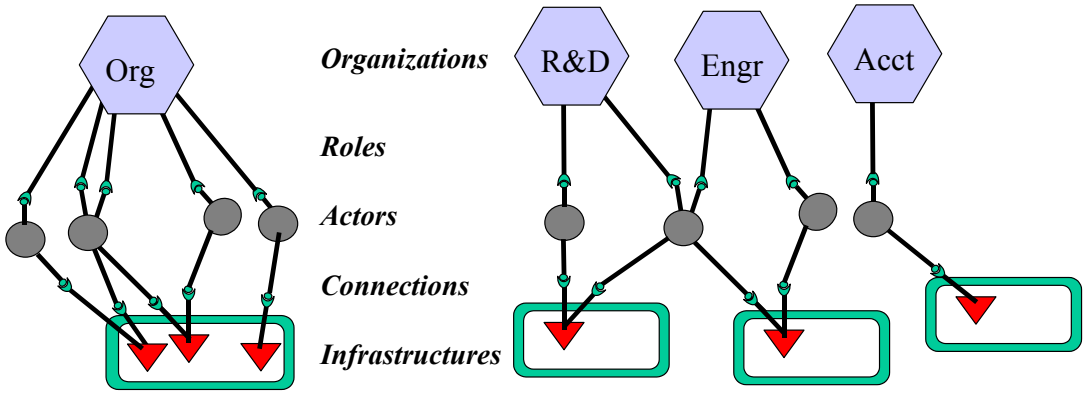


Figure 70. Example deaggregated organization.

2. Visualization of Large Societies

As the size of societies increase, the ability to present the society in a meaningful, visual way decreases. The analyst's screen becomes cluttered, and the ability to infer what is occurring in real-time decreases. This difficulty can make the analysis of the society very difficult, resulting in the analyst having to abandon the real-time interface and graphical notation of STIAM, and reverting to traditional analysis of reams of output or statistical analysis.

This lack of scalability in the current implementation of STIAM can be resolved by implementing a 'scenario recorder' that records the simulation run. This capability would permit the analysts to pause a simulation, and 'rewind' to examine what occurred at a particular point in time. This ability would also allow an analyst to 'record' a simulation, and examine the simulation graphically after execution. This is left as future work.

E. SUMMARY

This chapter demonstrates that scenarios found with the IA domain can be adequately simulated using a biologically based implementation of STIAM. This demonstrates that implementations of STIAM can be used as virtual IA laboratories to investigate portions of the IA domain that may not be easily observable. The observations and hypothesis generated from these scenarios are validateable with observations in the real environment.

The real strength of these scenarios is as a hypothesis generator, to cause security analysts to go to the real environment and seek answers to confirm or deny observations discovered in the implementation of the STIAM model. If an observation is confirmed, then these observations become theories to aid in the security analyst's understanding of the complex domain of IA. If the observations are denied, then the scenarios are adjusted and rerun to take into consideration elements and interaction that the researcher failed to consider previously. The procedure is repeated, causing a gradual increase in understanding of the IA environment.

IX. CONCLUSIONS AND RECOMMENDATIONS

This final chapter provides a summary of the major contributions of this dissertation. While this dissertation provides contributions to the field, it also raises new questions. Therefore, this chapter concludes with recommendations for future work.

A. CONCLUSIONS

STIAM provides a fundamental new approach to examining information assurance issues at an organizational level. The computational model provides a formal and descriptive notation for depicting the IA environment. Iconnectors provide a graphical notation that allows researchers to present the computational model in terms of a society in a connector notation, which aids in clarity. Iconnectors also provide a mechanism to implement graphical models as computational systems, and a communications mechanism to facilitate inter-entity interactions. The connector-based agent architecture provides researchers with composite agents constructed from relatively simple components that are capable of complex behavior.

The computational model and simulation permit researchers to select various levels of abstraction, and investigate particular properties in IA. This abstraction permits researchers to examine specific challenges in information assurance without extensive modeling of hardware and software details.

The proof-of-principle software architecture demonstrates the feasibility of this model. The components of the model capture the pertinent elements of the domain, and allow researchers to examine equivalence classes of vulnerabilities and exploits found in an environment, and implement a computational model of these as case studies.

Additionally, researchers can model the social interactions that are facilitated and constrained by technology. The model simulates the challenges in the domain; such as the inability to discover an agent's identity, location, means, or intent.

The computational model and simulation of the information security domain may provide valuable insight into current problems, as well as discover new challenges and solutions as they are revealed by the adaptive, evolutionary nature of the multi-agent system.

B. RECOMMENDATIONS FOR FUTURE WORK

Due to its modular design, STIAM provides a useful test bed for examining a variety of issues in information assurance, social and organizational modeling, and multi-agent system design. Below are some of the possible areas for additional research.

1. Agent History

Connector-based agents do not have an explicit history component. Rather, an implied history may be stored in the agent's tickets. History could be embedded in a connector-based agent using data objects, connectors, and frame pointers that may be stored in tickets. Actors could also retain a perception database to remember previous interactions with other agents, and the effect of actions on specific entities within the society. An important research area is the effect of agent histories on the behavior of agents in the STIAM system.

2. Behavior Moderators

The behavior moderators for STIAM agents were selected through a review of IA literature. A more thorough investigation of the moderators can be done, followed by research into how varying these values affect an agent and the society.

3. Dynamic Role Assignment Assignments and Organizations

In this implementation, agents are assigned roles and receive the role's component goals. A more dynamic implementation would allow actors to enter and exit roles and organizations throughout the simulation.

There are latency issues is dynamic role and organization assignments. After an actor leaves an organization, and the roles are broken, he still may have, or have the potential to have, bindings to infrastructures of that organization. These "ghost" or "phantom roles" demonstrate the danger of static tokens on security.

The duration and continuity of an organization may range from relatively static, such as incorporated conglomerates, to quite dynamic. Static organizations have to deal with actors whose goals change over time, and thus the organizations should adapt over time. Dynamic organizations force actors to adapt. STIAM is ideal for investigating dynamic organizations and actors.

4. Generating Tickets, Frames, and Actions at Runtime

Ticket sets provide a limited set of options to achieve a goal. An extension is to dynamically generate tickets and frames at runtime. This could be implemented as a genetic algorithm that tries new methods to achieve goals, and in effect investigates potentially innovative means to attack or defend entities in the pursuit of goals. Researchers may use this line of research to examine the coevolution of attackers and defenders.

5. Agent Learning

Agent learning was not implemented in the basic connector-based agents of STIAM. Learning, or autonomously improving an actor's behavior over time, may be implemented by modifying the weights to tickets and actions that have proven to be useful in the past, and throwing away tickets and actions that are not useful. This would provide an exciting advancement to STIAM. The basic research may result in the improvement of agent performance over time. A more interesting research area is in manipulating agents into learning a behavior, and then exploiting that behavior.

6. Complex Agent Goal Assignments

The actors that were implemented in the current version of the STIAM model had limited goals, and as such limited opportunities for internal conflict. More research needs to be conducted on the scalability of reactive, ticket-based agents and connector-based systems.

7. Discretionary Access Control Policies in the STIAM Model

In the current model, policies are static and cannot be changed by entities. Additionally, actors cannot grant or deny entities access to resources during execution of the STIAM model. Implementing the capability to modify agent access policies and interfaces during runtime would provide a tremendous improvement over enumerating all interfaces prior to runtime. This capability would represent a more dynamic environment and permit security analysts to accurately represent discretionary access control policies.

C. SUMMARY

This chapter presented the significant contributions of this dissertation. There are significant areas for future work – in both extending the model, and using the model and implementation for gaining insight into the IA domain. The connector-based simulation work provides a fruitful area of exploration, extending the insight gained from this dissertation into other research domains.

Chapter VI provided a validation that the elements found in the IA domain can be adequately represented in STIAM. Additionally, Chapter VIII demonstrates that an implementation of STIAM can generate scenarios and representable data that is found in the real world. Combined, these two results confirm the hypotheses of this dissertation.

LIST OF REFERENCES

Amori, R. D., "An Adversarial Plan Recognition System for Multi-Agent Airborne Threat," *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing,* vol. 1, pp 497-504, March 1 - 3, 1992.

Amoroso, E.G., *Fundamentals of Computer Security Technology*, Prentice-Hall Publishers, Upper Saddle River, NJ, 1994.

Anderson, E., *A Demonstration of the Subversion Threat: Facing the Critical Responsibility in the Defense of Cyberspace*, Masters Thesis, Naval Postgraduate School, Monterey, CA, March 2002.

Anderson, J.K., *Rules of the Mind*, Hillsdale, NJ, Lawrence Erlbaum, 1993.

Anderson, K., "Intelligence-Based Threat Assessments for Information Networks and Infrastructures," http://www.aracnet.com/~kea/Papers/threat-white-paper.shtml (4 Nov 1998).

Applegate, C., Elsaesser, C., and Sanborn, J., "An Architecture for Adversarial Planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no 1, January/February 1990.

Arbaugh, W.A., Fithen, W.L., and McHugh, J., "Windows of Vulnerability: A Case Study Analysis", *IEEE Computer*, vol. 22, no. 12, pp. 52-59, December 2000.

Arthur, B., "Inductive Reasoning and Bounded Rationality," *American Economic Association Papers,* vol. 84, pp. 406-411, 1994. http://www.santafe.edu/arthur/Papers/El_Farol.html (1 Sept 2001).

Atkins, D., Buis, P., Hare, C., Kelley, R., Nashenberg, C., Nelson, A. B., Phillips, P., Ritchey, T., and Steen, W., *Internet Security Professional Reference,* New Riders Publishing, IN, 1996.

Axelrod, R., *The Evolution of Cooperation*, Basic Books, 1984.

Axelrod, R., *The Complexity of Cooperation*, Princeton University Press, 1997.

Axtell, R., and Epstein, J.M., *Growing Artificial Societies: Social Science from the Bottom Up*, The Brookings Institute, Washington, D.C., 1996.

Barber, K.S., Liu, T.H., Goel, A., and Martin, C.E., "Conflict Representation and Classification in a Domain-Independent Conflict Management Framework," University of Texas – Austin, 1998.

Bell, D., and La Padula, L., "Secure Computer Systems: Mathematical Foundations and Model," *MITRE Report*, MTR-2547 vol. 2, November 1973.

Bellovin, S.M., "Security Problems in the TCP/IP Protocol Suite," *Computer Communications Review*, vol. 19, no. 2, pp 32-48, April 1989.

Biba, K., "Integrity Considerations for Secure Computer Systems," *U.S. Air Force Electronic Systems Division Technical Report,* 76-372, 1977.

Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

Brewer, D. and Nash, M., "The Chinese Wall Security Policy," Proceedings IEEE Symposium on Security and Privacy, IEEE Computer Society Press, pp 206-214, 1989.

Brinkley, D. L. and Schell, R. R., "Concepts and Terminology for Computer Security," ed. Abrams and Jajodia and Podell, *Information Security: an Integrated Collection of Essays*, IEEE Computer Society Press, Los Alamitos, CA, 1994.

Brinkley, D. L. and Schell, R. R., "What is There to Worry About? An Introduction to the Computer Security Problem," ed. Abrams and Jajodia and Podell, *Information Security: an Integrated Collection of Essays*, IEEE Computer Society Press, Los Alamitos, CA, 1994

Carley K. M. and Newell, A., "The Nature of the Social Actor," *Journal of Mathematical Sociology*, vol. 19(4), pp. 221-262, 1994.

Carroll, J. M. "A Portrait of a Computer Criminal," p42, *Information Security – The Next Decade,* Editors: J. H.P. Eloff and S. H. vonSolms, Chapman & Hall, UK, 1995.

Castelfranchi, C., Falcone, R., and de Rosis F., "Deceiving in GOLEM: How to Strategically Pilfer Help," *Autonomous Agent '98: Working notes of the Workshop on Deception, Fraud and Trust in Agent Societies*, 1998.

Clark, P.C., Supporting the Education of Information Assurance with a Laboratory Environment,*5th National Colloquium for Information System Security Education*, George Mason University, Fairfax, Virginia, May 2001

Computer Emergency Response Team (CERT), CERT Coordination Center, Software Engineering Institute, Carnegie Mellon, http://www.cert.org. (January 2002).

Cohen, F., "Computer Viruses: Theory and Experiments," *Computers and Security*, vol. 6, pp. 22-35, 1987.

Cohen, F., "Simulating Cyber Attacks, Defenses, and Consequences," http://all.net/journals/ntb/simulate/simulate.html, (7 Sept 2000).

Coveney, P. and Highfield, R., *Frontiers of Complexity: The Search for Order in a Chaotic World*, Fawcett Books, 1995.

Denning, D., "Concerning Hackers Who Break into Computer Systems," *Proceedings of the 13th National Computer Security Conference*, pp. 653-664, Washington, D.C., October 1-4, 1990.

Denning, D., *Information Warfare and Security*. Addison-Wesley Publishing, 1998.

Denning, D., Neumann, P., and Parker, D., "Social Aspects of Computer Security," in *Proceedings 10th National Computer Security Conference*, pp. 320-325, September 1987.

U.S. Department of Defense (DoD) CJCSI 3210.01, *Joint Information Warfare*, U.S. Department of Defense, January 1996.

U.S. Department of Defense (DoD) Joint Chiefs of Staff, *Information Assurance Through Defense in Depth*, U.S. Department of Defense, February 2000.

U.S. Department of Defense, Defense Information Systems Agency (DISA), *NETWARS,* U.S. Department of Defense, http://www.disa.mil/D8/netwars/about/index.htm, (November 2001).

Donaldson, T., "A Position Paper on Collaborative Deceit," http://citeseer.nj.nec.com/107418.html, (November 2000).

Dougherty, J.E. and Pfaltzgraff, R.L. Jr. *Contending Theories of International Relations*, Harper and Row, NY, 1981.

Donath, J., *Identity and Deception in the Virtual Community*, November 12, 1996, http://persona.www.media.mit.edu/judith/Identity/IdentityDeception.html, (Nov 2000).

Echo, *John Holland's Echo,* 2000, http://www.santafe.edu/projects/echo, (30 July 2000.)

Encyclopedia Britannica, http://www.britannica.com, (Dec 2000).

Ephrati, E., and Rosenschein, J., "Divide and Conquer in Multi-Agent Planning," Proceedings of the 12th National Conference on Artificial Intelligence, July-August, 1994.

Ferber, J., *Multi-Agent Systems, an Introduction to Distributed Artificial Intelligence*, Addison-Wesley Publishers, 1999.

Ferraiolo, D. and Kuhn, R., "Role-Based Access Control", *Proceedings of the 15th National Computer Security Conference*, pp. 554-563, Baltimore, MD, October 1992.

Filkes, R.E., Nilsson, N.J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence*, vol. 2, pp. 189-208, 1971.

Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T., "A Sense of Self for Unix Processes," *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, CA, pp. 120–128, 1996.

Geddes, N.D., "A model for intent interpretation for multiple agents with conflicts," *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, vol. 3, pp. 2080 –2085, October 2-5, 1994.

Geddes, N.D., "Large-scale models of cooperative and hostile intentions," *Proceedings of the 1997 workshop on Engineering of Computer-Based Systems*, pp. 142–147, 1997. http://computer.org/proceedings/ecbs/7889/78890142abs.htm, (Nov 2000).

Geddes, N.D., Smith, D.M., and Lizza, C.S. "Fostering collaboration in systems of systems," *1998 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 950–954, 1998.

Gmytrasiewicz, P.J., and Durfee, E.H., "Toward a Theory of Honesty and Trust Among Communicating Autonomous Agents," *Group Decision and Negotiation* 2:237-258, 1993.

Goguen, J. A., Meseguer, J. "Security Policies and Security Models," in *Proceedings of the 1982 IEEE Symposium of Security and Privacy*, pp. 11-20, April 1982.

Graham, R. and Denning, P. "Protection – Principles and Practices," *Proceedings AFIPS Spring Joint Computing Conference*, v. 40, pp. 417-429, 1972.

Taylor, P.A., *Hackers - Crime and the Digital Sublime*, Routledge, 1999.

Harrison, M., Ruzzo, W.L. and Ullman, J., "Protection in Operating Systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461-471, August 1976.

Hiles, J., VanPutte, M., Osborn, B., Zyda, M., *Innovations in Computer Generated Autonomy at the MOVES Institute*, Technical Report NPS-MV-02-002, Naval Postgraduate School, Monterey, California, 2001.

Hiles, J., Lewis, T., Osborn, B., Zyda, M., VanPutte, M., *StoryEngine: Dynamic Story Production Using Software Agents That Discover Plans*, Technical Report NPS-MV-02-TBA, Naval Postgraduate School, Monterey, California, 2002.

Hodges, J. and Dewar, J., *Is it You or Your Model Talking? A Framework for Model Validation*, R-4114-AF/A/OSD, RAND, Santa Monica, CA, 1992.

Holland, H., *Hidden Order – How Adaptation Builds Complexity*, Perseus Press, 1996.

The Honeypot Project (ed), *Know Your Enemy – Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*, Addison-Wesley, 2002.

Horstmann, C., Cornell, G., *Core Java 2, Volume 1: Fundamentals 5th edition*, Prentice Hall Publishers, Upper Saddle River, NJ, December 2000.

Howard, J.D., *An Analysis of Security Incidents on the Internet, 1989-1995*, Engineering and Public Policy Dissertation, Carnegie-Mellon University, April 7, 1997.

Howard, J.D. and Longstaff, T.A., *A Common Language for Computer Security Incidents,* (SAND98-8667), Livermore, CA: Sandia National Laboratories, 1998. http://www.cert.org/research/taxonomy_988667.pdf, (Mar 2002).

Ilachinski, A. *Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare,* Center for Naval Analysis Research Memorandum CRM 97-61.10 August 1997, Center for Naval Analysis, Alexandria, VA, 1997.

Irvine C., and Levin, T., "Teaching Security Engineering Principles," *Proceedings Second World Conference on Information Security Education*, Perth, Australia, pp. 113-127, July 2001.

Irvine, C. E., "The Reference Monitor Concept as a Unifying Principle in Computer Security Education," *Proceedings of the First World Conference on Information Systems Security Education*, Stockholm, Sweden, pp.27-37, June 1999.

Irvine, C. E., "Amplifying Security Education in the Laboratory," *Proceedings of the First World Conference on Information Systems Security Education*, Stockholm, Sweden, pp. 139-146, June 1999.

Irvine, C. E., Warren, D.F., and Clark, P.C., "The NPS CISR Graduate Program in INFOSEC: Six Years of Experience," *Proceedings of the 20th National Information Systems Security Conference*, Baltimore, MD, pp.22-30, October 1997.

Irvine, C. E., Chin, S., and Frinke, D., "Integrating Security into the Curriculum", *IEEE Computer*, vol. 31, no. 12, , pp.25-30, 1998.

Jones, R. M., Laird J.E., Nielsen P.E. Coulter, K.J., Kenny P.G. and Koss F.V., "Automated Intelligent Pilots for Combat Flight Simulation," *AI Magazine*, vol. 20(1), pp. 27-42, 1999.

Kang, M., Waisel, L.B., Wallace, W.A. "Team Soar – A Model for Team Decision Making," *Simulating Organizations*, AAAI Press, 1998.

Karger, P. A. and Schell, R. R., *Multics Security Evaluation: Vulnerability Analysis*, ESD-TR-74-193, Vol. II, Headquarters Electronic Systems Division, Hanscom Air Force Base, MA, June 1974.

Katzela, I., *Modeling and Simulating Communications Networks: A Hands-on Approach Using OPNET*, Prentice Hall, 1998.

Klein, G., *Source of Power, How People Make Decision*, MIT Press, 1999.

Krsul, I. V., *Software Vulnerability Analysis*, Ph.D. Dissertation, Computer Sciences Department, Purdue University, Lafayette, IN, May 1998.

Laird, J.E., Newell, A., Rosenbloom, P.S., "SOAR: An Architecture for General Intelligence", *Artificial Intelligence*, vol. 33, pp. 1-64, 1987.

Landwehr C.E., Bull, A.R., McDermott, J.P., and Choi, W.S., "A Taxonomy of Computer Security Flaws," *ACM Computing Surveys*, vol. 26, no. 3, pp. 211-254, September 1994.

Langton C., *Artificial Life*, Addison-Wesley, 1988.

Langton, C. (Ed), *Artificial Life: An Overview*, The MIT Press, Cambridge, MA, 1997.

Law, W. and Kelton, W., *Simulation Modeling and Analysis*, McGraw Hill, 2000.

Liu, L., Yu, E., Mylopoulos, J., *Analyzing Security Requirements among Strategic Actors*, to appear at *Second Symposium on Requirements Engineering for Information Security*, Raleigh, North Carolina, October 2002.

Lunt, T., "Access Control Policies for Database Systems," *Database Security, II: Status and Prospects -- Result of the IFIP WG 11.3 Workshop on Database Security*, Kingston, Ontario, Canada, 5-7 October, 1988, North-Holland, pp. 41-52, 1988.

Machiavelli, N., *The Prince*, Bantam Classics Publishing, (reissue Sept 1984), 1515.

Meritt, J.W., "A Method for Quantitative Risk Analysis," *22nd National Information System Security Conference*, Arlington, VA, October, 1999.

Minehart, R., *The Information Assurance Seminar Game*, Center for Strategic Leadership, U.S. Army War College, Carlisle Barracks, PA, 1998.

Myers, P., A., *Subversion: The Neglected Aspect of Computer Security*, Masters Thesis, Naval Postgraduate School, Monterey, California, June 1980.

National Institute of Standards and Technology (NIST), U.S. Department of Commerce, 1993-10-04 "Glossary of Computer Security Terms" Version 1, 10/21/88 - Rainbow Series, http://csrc.ncsl.nist.gov/secpubs/rainbow/tg004.txt, (March 2002).

National Institute of Standards and Technology (NIST), U.S. Department of Commerce, *An Introduction to Computer Security: The NIST Handbook*, Special Publication 800-12, 1996.

National Research Council (NRC), *Modeling Human and Organizational Behavior*, National Academy Press, Washington D.C., 1998.

National Security Telecommunications and Information System Security Committee (NSTISSC), *NSTISSI No. 4009 - National Information Systems Security (INFOSEC) Glossary*, September 2000.

Nelson, P., *The Penguin Dictionary of Mathematics*, David Nelson (ed), 2nd ed., Penguin Press, London, 1998.

Neumann, P. and Parker, D., "A Summary of Computer Misuse Techniques," In *Proceedings of the 12th National Computer Security Conference*, pages 396–407, Baltimore, Maryland, USA, Oct. 10–13, 1989.

Neumann, P., *Computer-Related Risks*, ACM Press and Addison-Wesley, 1995.

Newell, A., and Simon, H.A., "GPS – A Program that Simulates Human Thought," in *Computers and Thought*, Feigenbaugh E.A. and Feldman, J., eds., McGraw-Hill Publishing, New York, New York, 1963.

Osborn, B., *"An Agent-Based Architecture for Guiding Interactive Stories,"* Ph.D. Dissertation, U.S. Naval Postgraduate School, Monterey, California, expected completion September 2002.

Parker, D., *Fighting Computer Crime*, John Wiley & Sons, 1998.

Picault S. and Collinot A., "Designing Social Cognition Models for Multi-Agent Systems Through Simulating Primate Societies," *Proceedings of the Third International Conference on Multi-Agent Systems* (ICMAS'98), IEEE Press, 1998.

Pfleeger, C., *Security in Computing*, Prentice-Hall Publishing, 1997.

Pnueli, A., Specification and Development of Reactive Systems, In *Information Processing'86,* pp. 845-858, Elsevier Press, North Holland, 1986.

Prietula, M. J., Carley, K.M., Gasser, L., "A Computational Approach to Organizations and Organizing," *Simulating Organizations*, AAAI Press, 1998.

Raskin, V., Nirenburg, S., "Ontology in Information Security: A Useful Theoretical Foundation and Methodological Tool," *Proceedings of the New Security Paradigm Workshop 2001*, September 2001, Cloudcroft, NM, 2001.

Rheingold, H., *The Virtual Community, Homesteading on the Electronic Frontier*. Addison-Wesley Publishing, 1993.

Roddy, K.A. and Dickson, M.R. *"Modeling Human Organizational Behavior Using a Relation-Centric Multi-Agent Design Paradigm,"* Master's Thesis, U.S. Naval Postgraduate School, Monterey, California, Sept 2000.

Rowe, N. and Schiavo, S., "An Intelligent Tutor for Intrusion Detection on Computer Systems", *Computers and Education*, vol. 31, pp. 395-404, 1998.

Rumbaugh, J., Jacobson, I., Booch, G., *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

Russell, D., and Gangemi, G.T., *Computer Security Basics,* O'Reilly & Associates, Sebastopol, CA, 1991.

Russell, S. and Norvig, P., *Artificial Intelligence, A Modern Approach,* Prentice Hall, 1995.

Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces", *Artificial Intelligence*, vol. 5, pp. 115-135, 1974.

Schillo, M. and Funk, P.*, "*Learning From And About Other Agents In Terms Of Social Metaphors," *Proceedings of the ``Agents Learning About, From and With Other Agents' Workshop*, 1999, http://jmvidal.ece.sc.edu/alaa99/schillo.ps, (Jan 2001).

Schneier, B., *Managed Security Monitoring: Closing the Window of Exposure*, http://www.counterpane.com/window.html, (Mar 2002).

Schwartau, W., *Information Warfare-Cyberterrorism: Protecting your Personal Security in the Electronic Age*, Thundermouth Press, 1994.

Schwartau, W., *Information Warfare-Chaos on the Electronic Superhighway*, Thundermouth Press, 1996.

Schwartau, W., *Time Based Security*, Interpact Press, 1999.

Slatalla, M., and Quittner, J., *Masters of Deception – The Gang that Ruled Cyberspace*, Harper Collins Publishing, 1995.

Steele, G., Woods, D., Finkel, R., Crispin, R., Stallman, R., Goodfellow, G., *The Hacker's Dictionary*, Harper & Row, NewYork, 1983.

Sterne, D. F., On the Buzzword "Security Policy," *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1991.

Stoll, C., *The Cuckoo's Egg – Tracking a Spy Through the Maze of Computer Espionage*, Pocket Books, 1990.

Summers, R.C., *Secure Computing – Threats and Safeguards*, McGraw-Hill, 1997.

Sussman, G.J., "The Virtuous Nature of Bugs", ed. Allen, J., Hendler, J., Tate, A., *Readings in Planning*, Morgan Kaufmann Publishers, 1990.

Tecuci G., Hieb, M.R., Hille, D. and Pullen, J.M., "Building Adaptive Autonomous Agents for Adversarial Domains," *Proceedings of the AAAI Fall Symposium on Planning and Learning*, November 1994.

Thompson, K., "Reflections on Trusting Trust," *Communications of the ACM,* vol. 27, no. 8, pp. 761-763, August 1984.

VanPutte, M., Osborn, B., Hiles, J., A Composite Agent Architecture for Multi-Agent Simulations, *Proceedings of the Eleventh Conference in Computer Generated Forces and Behavior Representation*, May 2002.

Von Clausewitz, C., (eds) Howard, M., and Paret, P., *On War*, Princeton University Press, 1989.

Wadlow, T.A., The *Process of Network Security*, Addison-Wesley, 2000.

Wagner, T., Shapiro, J., Xuan, P. and Lesser, V., "Multi-Level Conflict in Multi-Agent Systems," *Proceedings of the 1999 AAAI Workshop on Negotiations in Multi-Agent Systems,* 1999, http://www-net.cs.umass.edu/~jshapiro/, (Nov 2000).

Walker, J., "Unsafe at Any Key Size: An Analysis of the WEP Encapsulation," Technical Report 03628E, IEEE 802.11 Committee, October 2000. http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/0-362.zip, (March 2002).

Weiss, G., (ed), *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence,* MIT Press, 1999.

Wellman, M. P., entry for *The MIT Encyclopedia of the Cognitive Sciences,* http://vulture.eecs.umich.edu/faculty/wellman/pubs/multiagent-ECS.text, (Oct 2000).

White, H., "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, vol. 1, pp. 425-464, 1989.

Wildberger, A. M., "AI & Simulations," *Simulations (Magazine)*, pp. 171, September 2000.

Willmott, S., Bundy A., Levine, J., and Richardson, J., *Adversarial Planning in Complex* Domains, http//ailab.hyungpook.ac.kr/Seminar/20000216/fullpaper.html, (Nov 2000).

Winkler, I., *Corporate Espionage: What it is, Why it is Happening in Your Company, and What You Must Do About it*, Prima Publishing, 1997.

Wooldridge, M., and Jennings, N.R., "Intelligent Agents: Theory and Practice," *The Knowledge Engineering Review,* vol. 10(2), pp. 115-152, 1995.

Wooldridge, M. *Reasoning about Rational Agents*. MIT Press, July 2000.

Yu, E. Towards "Modelling and Reasoning Support for Early-Phase Requirements Engineering", *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, Washington D.C., pp. 226-235, January 6-8, 1997.

GLOSSARY

actor – a synthetic representation of IA relevant people who interact in the environment and are therefore represented in the society.

agent – an active entity in the society generally representing a person or autonomous process.

attack – "A series of steps taken by an attacker to achieve an unauthorized result" [Howard and Longstaff, 1998].

attacker – An actor who attempts to achieve an unauthorized result[16].

authenticate – To verify the identity of a user, user device, or other entity, or the integrity of data stored, transmitted, or otherwise exposed to unauthorized modification in an information system, or to establish the validity of a transmission [NSTISSC, 2000].

availability – "Timely, reliable access to data and information services" [NSTISSC, 2000]. See confidentiality, integrity.

confidentiality – "Assurance that information is not disclosed to unauthorized persons, processes or devices" [NSTISSC, 2000]. See integrity, availability.

control/countermeasure – those things which are implemented to prevent exposure to the threat in the first place, detect if the threat has been realized against the system, mitigate the impact of the threat against the system, or recover/restore the system [Meritt, 1999] .

denial of service – A type of incident resulting from any action or series of actions that prevents any part of an information system from functioning [NSTISSC, 2000].

distributed attack – typically has a 'master' who centrally controls multiple 'zombies' on compromised hosts. At the direction of the master, the zombies perform a coordinated attack against a designated 'target' host.

entity – any element in the set of actors, organizations, or infrastructures.

environment – a real world situation or system being modeled. A society is a highly abstract representation of a particular environment.

exploit – Events that occur that cause undesirable consequences on the part of the victim. Actors possess exploits, and use these against infrastructures and other actors in the hope of exploiting a vulnerability.

[16] Adapted from [Howard and Longstaff, 1998].

firewall – an access control mechanism that is designed to defend against unauthorized access to or from a private network [NSTISSC, 2000].

hardening system – installing patches and removing unused system services in order to eliminate vulnerabilities from a system.

information assurance (IA) – "…protect(ing) and defend(ing) information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation" [NSTISSC, 2000]

information resource – see resource.

information warfare – "Actions taken to achieve information superiority by affecting adversary information, information-based processes, information systems, and computer-based networks while defending one's own information, information-based processes, information systems, and computer-based networks [DoD, 1996]

infrastructure – the key information and information systems that exist for an organization.

integrity – "…protection against unauthorized modification or destruction of data (and processes)". [NSTISSC, 2000]. See confidentiality, availability.

logical attack – refers to manipulating data in an electronic format. See physical attack and social attack.

object – A passive entity that exists in the society. See agent.

organization – an abstract representation of social entities that exist for a particular purpose.

patch-and-penetrate – technique used in the 1970s and 1980s in the hopes of building a trustworthy IT system. It consisted of patching known vulnerabilities in a system, then breaking into the system again, and reiterating until the engineers could no longer break in.

penetration – "the successful act of bypassing the security mechanisms of the system" [NIST, 1988] in order to gain access past the security protection.

physical attack – refers to the theft, destruction, and/or damage of materials. See social engineering and logical attack.

policy/security policy – a set of rules specified by an organization that describe who may access a certain resource, and for what purpose.

process – a computer program in execution [NIST, 1988].

protocols – A series of steps taken by two or more parties to accomplish same task.

resource – critical information or processes whose confidentiality, integrity, or availability is required for an organization to exist.

risk – the possibly that a particular threat will adversely impact an information system by exploiting a particular vulnerability [NSTISSC, 2000].

script kiddie – "…(a person) with limited technical expertise using easy-to-operate, pre-configured, and/or automated tools to conduct disruptive activities against networked systems" [Steele, 1983].

simulation – a method, usually involving hardware and software, for implementing a model to play out the represented behavior over time [NRC, 1998].

social engineering – using nontechnical interpersonal deception to manipulate individuals into providing information in order to bypass security controls. Also referred to as *perception management*. See physical attack and logical attack.

society – an abstract representation of the critical entities, structures, and relationships found in an environment. A society is comprised of sets of organizations, infrastructures, and actors.

spoof – "An active security attack in which a machine on the network masquerades as a different machine" [Howard and Longstaff, 1998].

subversion – the "covert and methodical undermining of internal and external controls over a system lifetime to allow unauthorized and undetected access to system resources and/or information." [Myers, 1980].

threat – Any circumstance or event with the potential to cause harm to a system in the form of destruction, disclosure, modification of data and/or denial of service [NIST, 1988]

token – an abstract representation of static objects that are found in the environment being modeled.

Trojan horse – a small piece of malicious code hidden within an attractive legitimate program.

user (end user) – an actor for whom information systems are developed.

vulnerabilities – a weakness in an entity allowing actions that are undesirable for legitimate users.

worm – autonomous self-replicating program that spreads from one system to another exploiting holes in the system

zombie – a compromised host computer that has functionality added by an attacker that allows the attacker (master) to control the host. Typically, masters will send orders to zombies to attack other machines, as in denial of service attacks.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – EXECUTION OUTPUT

This appendix is provided to allow a detailed analysis of the results of Scenario One. It begins with the loading of the scenario into the simulation. Next, it runs in chronological order, listing the goals and actions of each actor. The output is halted after the attacker successfully accesses the resource on the enterprise infrastructure.

SCENARIO ONE –ADAPTIVE ATTACKER

// LOADING THE SCENARIO INTO THE SIMULATION ENGINE

```
creating new IBinder: environment          // ibinder named 'environment' is created
Adding new Token: enterprise               // all tokens are registered in environment
Adding new Token: enterpriseService
Adding new Token: sysType
Adding new Token: vuln103
Adding new Token: dbPassword
Adding new Token: malice
IConnector Changed -- iconnector(library)  // begin constructing library infrastructure
   library  extended(true)
Infrastructure added: 0: Infrastructure library    // library infrastructure created
IConnector Changed -- iconnector(hackerSite)       // begin constructing hacker
  infrastructure
   hackerSite  extended(true)
Infrastructure added: 2: Infrastructure hackersite// hacker infrastructure created
IConnector Changed -- iconnector(enterprise)       // begin constructing enterprise
  infrastructure
   enterprise  extended(true)
IConnector Changed -- iconnector(enterprise)
   enterprise  extended(true)
Adding new resource database
IConnector Changed -- iconnector(database)
   database  extended(true)
Infrastructure added: 4: Infrastructure enterprise// enterprise infrastructure created
Hacker: hacker received message: message:          // begin creating hacker actor
     from: null
    token: Token: enterprise
    memo: initial token
Hacker: hacker added token: Token: enterprise      // hacker received 'enterprise' token
Actor added:  Hacker: hacker                       //hacker created
```

//SIMULATION BEGINS

```
*** Clock time now: 1
Hacker: hacker executing goal Goal: GatherIntelGoal
  executing frame: Action: ConductLibraryResearchAction
IConnector Changed -- iconnector(library)
   library  extended(true)
  Socket Binding -- hacker to library
Hacker: hacker received message: message:
     from: 0: Infrastructure library
    token: Token: enterpriseService
    memo: receive information on enterprise
Hacker: hacker added token: Token: enterpriseService
IConnector Changed -- iconnector(library)
library     extended(false)
library  disconnecting from Socket: library

*** Clock time now: 2
Hacker: hacker executing goal Goal: GatherIntelGoal
  executing frame: Action: ConductLibraryResearchAction
IConnector Changed -- iconnector(library)
   library  extended(true)
IConnector Changed -- iconnector(library)
library     extended(false)

*** Clock time now: 3
```

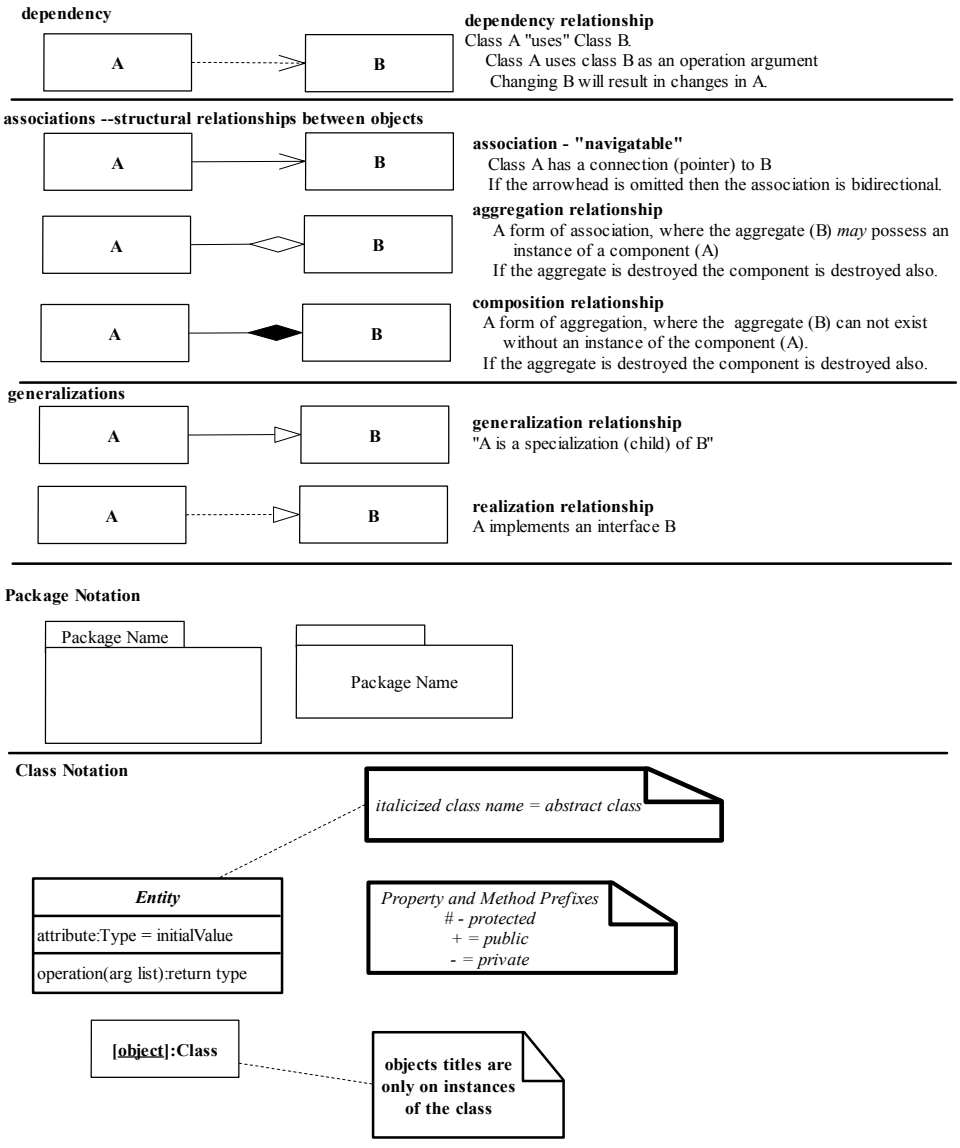
```
Hacker: hacker executing goal Goal: GatherIntelGoal
  executing frame: Action: ScanEnterpriseWithDataAction
IConnector Changed -- iconnector(enterprise)
   enterprise  extended(true)
IConnector Changed -- iconnector(enterprise)
enterprise     extended(false)
IConnector Changed -- iconnector(enterprise)
   enterprise  extended(true)
  Socket Binding -- hacker to enterprise
Hacker: hacker received message: message:
     from: 4: Infrastructure enterprise
    token: Token: sysType
    memo: receive detailed information on systems running on environment
Hacker: hacker added token: Token: sysType
IConnector Changed -- iconnector(enterprise)
enterprise     extended(false)
enterprise  disconnecting from Socket: enterprise

*** Clock time now: 4
Hacker: hacker executing goal Goal: GatherIntelGoal
  executing frame: Action: ConductLibraryResearchAction
IConnector Changed -- iconnector(library)
   library  extended(true)
IConnector Changed -- iconnector(library)
library     extended(false)

*** Clock time now: 5
Hacker: hacker executing goal Goal: GatherIntelGoal
  executing frame: Action: ScanEnterpriseWithDataAction
IConnector Changed -- iconnector(enterprise)
   enterprise  extended(true)
IConnector Changed -- iconnector(enterprise)
enterprise     extended(false)

*** Clock time now: 6
Hacker: hacker executing goal Goal: GatherIntelGoal
  executing frame: Action: ResearchSysVulnAction
IConnector Changed -- iconnector(hackerSite)
   hackerSite  extended(true)
  Socket Binding -- hacker to hackersite
Hacker: hacker received message: message:
     from: 2: Infrastructure hackersite
    token: Token: vuln103
    memo: receive exploit for 'vuln103' on system 'systype'
Hacker: hacker added token: Token: vuln103
IConnector Changed -- iconnector(hackerSite)
hackerSite     extended(false)
hackerSite  disconnecting from Socket: hackerSite

*** Clock time now: 7
Hacker: hacker executing goal Goal: ExpandPowerbaseGoal
  executing frame: Action: ExploitSysAction
IConnector Changed -- iconnector(enterprise)
   enterprise  extended(true)
  Socket Binding -- hacker to enterprise
Hacker: hacker received message: message:
     from: 4: Infrastructure enterprise
    token: Token: dbPassword
    memo: receive password to access Resource:database
Hacker: hacker added token: Token: dbPassword
IConnector Changed -- iconnector(enterprise)
enterprise     extended(false)
enterprise  disconnecting from Socket: enterprise

*** Clock time now: 8
Hacker: hacker executing goal Goal: EarnFameGoal
  executing frame: Action: AccessResourceAction
IConnector Changed -- iconnector(database)
   database  extended(true)
  Socket Binding -- hacker to enterprise
IConnector Changed -- iconnector(database)
database     extended(false)
database  disconnecting from ResourceSocket: database

*** Clock time now: 9
Hacker: hacker executing goal Goal: GatherIntelGoal
  executing frame: Action: ConductLibraryResearchAction
IConnector Changed -- iconnector(library)
```

```
   library  extended(true)
IConnector Changed -- iconnector(library)
library     extended(false)
IConnector Changed -- iconnector(library)
   library  extended(true)
IConnector Changed -- iconnector(library)
library     extended(false)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B –UML QUICK REFERENCE

This appendix summarizes the graphical notation for elements of the Unified Modeling Language (UML) that are used in this dissertation. See [Rumbaugh *et al.*, 1999] for a comprehensive reference manual.



THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Research Office, Code 09
   Naval Postgraduate School
   Monterey, CA

4. Dr. Michael Zyda
   Director, MOVES Institute
   Naval Postgraduate School
   Monterey, CA

5. Dr. Cynthia E. Irvine
   Computer Science Department
   Code CS/IC
   Naval Postgraduate School
   Monterey, CA 93943

6. Professor John Hiles
   MOVES Institute
   Naval Postgraduate School
   Monterey, CA

7. Dr. Don Brutzman
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA

8. Dr. Rudy Darken
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA

9. Dr. Neil Rowe
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA

10. Major Michael VanPutte
    MOVES Institute
    Naval Postgraduate School
    Monterey, CA

11. Commander, Naval Security Group Command
Naval Security Group Headquarters
9800 Savage Road
Suite 6585
Fort Meade, MD 20755-6585
San Diego, CA 92110-3127

12. James P. Anderson
James P. Anderson Co.
140 Morris Drive
Ambler, PA 19002

13. Dr. Mike Bailey - Director
Marine Corps Combat Development Command (MCCDC)
Doctrine Division (C42)
3300 Russell Road
Quantico, VA 22134-5001

14. Philip Barry, Ph.D.
Associate Technology Area Manager
Intelligent Information Management and Exploitation
The MITRE Corporation
1820 Dolley Madison Blvd. MS W640
McLean, Virginia 22102 USA

15. CAPT Richard Bump
Director, N6M
Navy Modeling & Simulation Management Office (NAVMSMO)
2000 Navy Pentagon PT 5486
Washington, DC 20350-2000

16. George Bieber
Human Resources and Training Division Chief
Defense-wide Information Assurance Program
Office of the Secretary of Defense

17. Ms. Elaine S. Cassara
Branch Head, Information Assurance
United States Marine Corps
HQMC, C4
2 Navy Annex
Washington, DC 20380

18. Ms. Louise Davidson,
N614
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

19. Mr. William Dawson
Community CIO Office
Washington DC 20505

20. LCDR Scott Dipert
N614
Presidential Tower
2511 South Jefferson Davis Highway
Arlington, VA 22202

21. Mr. Richard Hale
Defense Information Systems Agency, Suite 400
5600 Columbia Pike
Falls Church, VA 22041-3230

22. Dr. Harold L. Hawkins
Program Officer
Office of Naval Research Code 342
800 N. Quincy St.
Arlington, VA
22217-5660

23. Maj David Laflam
HQDA, DCS G3, (DAMO-ZS)
400 Army Pentagon
Washington, DC 20310-0400

24. CAPT Mike Lilienthal, USN - DMSO
Defense Modeling & Simulation Office
1901 N. Beauregard Street, Suite 500
Alexandria, Virginia, 22311-1705, USA

25. W. H. (Dell) Lunceford, Jr
Director, Army Model and Simulation Office
HQDA, DCS G3, (DAMO-ZS)
400 Army Pentagon
Washington, DC 20310-0400

26. Dr. Douglas Maughan
DARPA
3710 Fairfax Avenue
Arlington, VA 22203

27. VADM Richard Mayo
CNO, N6
2000 Navy Pentagon
Washington, DC 20350-2000

28. CAPT Dennis McBride, USN (ret)
VP, Potomac Institute
Potomac Institute for Policy Studies
901 N. Stuart Street, Suite 200,
Arlington, VA 22203

29. Ms. Deborah Phillips
Community Management Staff
Community CIO Office
Washington DC 20505

30. LTC John Quigg
Chief, NSIP
HQDA(SAIS-IAS)
2511 Jefferson Davis Hwy
Arlington, VA 22202

31. LCDR Dylan Schmorrow, USN
DARPA / IPTO
3701 Fairfax Drive
Arlington, VA 22203-1714

32. Dr. Randall Shumaker
Director, University of Central Florida, Institute for Simulation & Training
University of Central Florida
3280 Progress Drive, Orlando, FL 32826

33. Dr. Ralph Wachter
Office of Naval Research
Ballston Tower One
800 North Quincy Street
Arlington, VA 22217-5660

34. Jim Weatherly
Deputy Director NAVMSMO
2000 Navy Pentagon
Washington, D.C. 20350-2000

35. CAPT Robert A. Zellman
CNO N6
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202